CUT: Community Update and Tracking in Dynamic Social Networks *

Hao-Shang Ma
National Cheng Kung University
No.1, University Rd., East Dist., Tainan City , Taiwan
ablove904@gmail.com

Jen-Wei Huang
National Cheng Kung University
No.1, University Rd., East Dist., Tainan City , Taiwan
jwhuang@mail.ncku.edu.tw

ABSTRACT

Social network exhibits a special property: community structure. The community detection on a social network is like clustering on a graph, but the nodes in social network has unique name and the edges has some special properties like friendship, common interest. There have been many clustering methods can be used to detect the community structure on a static network. But in real-world, the social networks are usually dynamic, and the community structures always change over time. We propose Community Update and Tracking algorithm, CUT, to efficiently update and track the community structure algorithm in dynamic social networks. When the social network has some variations in different timestamps, we track the seeds of community and update the community structure instead of recalculating all nodes and edges in the network. The seeds of community is the base of community, we find some nodes which connected together tightly, and these nodes probably become communities. Therefore, our approach can quickly and efficiently update the community structure.

Categories and Subject Descriptors

H.3.4 [**Information systems applications**]: Data mining—*Clustering*; J.3.2 [**Collaborative and social computing**]: Collaborative and social computing design and evaluation methods—*Social network analysis*

General Terms

Theory

Keywords

Social Network Analysis, Community Detection, Dynamic Algorithm

1. INTRODUCTION

*SNAKDD'13, August 11 2013, Chicago, IL, USA Copyright 2013 ACM 978-1-4503-2330-7/13/08$15.00.

Social networks could be modeled as graphs with a set of nodes and edges. Each node represents an individual and the edge between two nodes represent the relationship or interaction of these nodes. In addition, social networks exhibit a very special property, the community structure [1]. Members in the same community of a social network usually have more common relationships such as interests, friends, topics of discussion or topics of research. Members usually interact with members in the same community more frequently than with members outside of their community. The detection of communities in a network usually chooses an objective function to gather network nodes into groups in such a way that nodes in the same group are densely connected to one another. Different objective functions may lead to different sizes and numbers of communities in a network. Knowing the community structure could help us to understand the properties of a network. For example, a community in a coauthorship network is the set of authors sharing the similar research topic.

One of challenges associated with community detection is the dynamic network. In fact, the real-world social networks are usually not static networks. The social network usually changes its own structure with the time. The relationship between two nodes may change or disappear, and some nodes may join or leave the social network. Consequently, the communities are altered with the variations of the social network over time. The community may grow, shrink, split or merge. When some nodes join or leave a community, the community grows or shrinks. If a big community becomes two or more small communities, the big one splits. Those small communities can also merge into a big one. Generally, we can know the community dynamics by tracking communities and identifying community representatives [2] [4] [17]. There are many research proposed for detecting communities in a static network [5] [8] [11]. Even though we can use their methods to find new community structures in a dynamic social network, any change of the social network will lead to recalculation of relationships between nodes and edges. However, the consequent temporal variation of nodes or edges in a dynamic network is usually small. If we could only update the communities of a social network in an efficient way instead of recalculating the properties of whole network, it would save lots of time. For this reason, we have to not only detect communities but also need to track them in order to update the changes efficiently. In this work, we propose a Community Update and Tracking, abbreviated as CUT, algorithm to track the seed of community in a dynamic social network, and deal with the net-

work variations to update communities at successive timestamps. The seed of community is a set of nodes which are connected to one another very tightly, and can be further extended to be a larger community. It is more efficient to track seeds of community than the whole nodes in communities. Based on the clique percolation method, we define the seed of community as a set of adjacent k-cliques (k equals 3)in the social network graph. If a 3-clique is shared some edges with another clique, these two cliques are said to be adjacent. In order to track seeds of community easily and efficiently, we transform the original social network containing nodes and edges into a bipartite graph containing information of cliques and the connectivity of those cliques. We give an serial number to each 3-clique in original graph and transform the number to bipartite graph as nodes in a disjoint set. Then, we put edges as nodes in another disjoint set in the bipartite graph. In the bipartite graph, each clique links to three edges. CUT is able to efficiently find the seeds of community by traversing the connecting components in that bipartite graph. When network changing with the time, if there are some nodes/edges added or removed, we can update them in the bipartite graph and obtain new seeds of community very quickly. Instead of tracking all communities, CUT only monitors seeds of community, which makes CUT more efficiently. Then, the communities in the dynamic network are constructed by expanding the seeds of communities.

The contributions of this work are as follows

1. We define the seeds of community to represent the core of communities. The seeds of community are most important parts of communities.
2. We propose a clique bipartite graph, CAB, to represent the relations of the seeds of community.
3. We design an efficient algorithm, CUT, to update and track the communities in dynamic social networks.

The experimental results show the advantages of CUT. CUT can efficiently update the communities and identify the community structure well. In addition, CUT has particularly prominent advantage when dealing with large variations of the network.

The rest of this work is organized as follows. In Section 2, a brief survey of related works is presented. The proposed algorithm is introduced in Section 3. Section 4 describes the experimental results. Finally, the work is concluded in Section 5.

2. RELATED WORKS

2.1 Static Community Detection

There are several methods proposed to detect the community structure in static social networks. Girvan [11] proposed a community discovery algorithm based on the iterative removal of edges with high betweenness scores. Following this, to reduce the computational cost of the betweenness-based algorithm, Newman [12] proposed a fast greedy modularity optimization method. The modularity is a measurement function to evaluate the overall quality of a graph partition.

Some community detection methods are based on finding cliques which are complete connected sub-graphs. Palla [8] proposed a clique percolation method, CPM. Communities are built up from k-cliques. Two k-cliques are said to be adjacent if they share k-1 nodes, and a k-clique community corresponds to a set of k-cliques in which they are all adjacent to one another. Yang [19] proposed ComTector algorithm to find all maximal cliques. Then, ComTector converts maximal cliques to initial partitions and assigns the rest vertices to the most appropriate partition. Yan [18] proposed an algorithm to detect disjoint cliques and then merged these cliques into communities to optimize the modularity.

2.2 Tracking Community Dynamics

Tracking community deviations can discover interesting behaviors in evolutionary network and understand the evolution of community with time. Tantipathananandh [16] proposed a framework for identifying communities by temporal changes. Kumar et al. [10] presented a model of network growth to capture singletons, isolated communities, and a giant component. Chen [4] proposed a method to detect community dynamics based on graph representatives and community representatives. Wang [17] presented a core-based algorithm of tracking community evolution by finding the most important nodes in a community to represent that community. Palla [14] used the k-clique percolation technique to investigate the time dependence of communities on a large scale social network.

2.3 Dynamic Community Detection

The problem of detecting communities in a dynamic social network has received much attention. The methods in this field have a common characteristic. They do not want to calculate all nodes and edges in the evolutionary network at each timestamp. Instead, updating existing community structures to new community structures is the most important issue. Cazabet [3] proposed a iLCD algorithm to update the existing community by two scores. Gregory [9] proposed a label propagation algorithm, COPRA. Each vertex updates its community by computing a belonging coefficient and the coefficient is the average coefficient from all of its neighbors. Nguyen [13] presented QCA algorithm based on tracing the community structure and processing the network changes. QCA uses the current community structure, and deal with the changing cases, new nodes, new edges, nodes removed, and edges removed to maximize the modularity. In QCA algorithm, they keep whole community structures to next timestamp. QCA needs to execute the procedure of CPM to obtain the remaining communities when every remove case. And CPM costs too much time at each updating procedure.

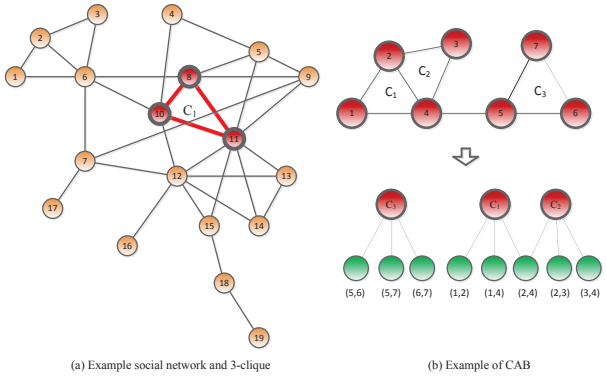
3. COMMUNITY UPDATE AND TRACKING ALGORITHM

In this section, we will illustrate the step of CUT algorithm. Given G = (V, E) an undirected graph with N nodes and M edges. V is the set of nodes in a graph. E is the set of edges in a graph. In social networks each node represents a unique person or item. Two nodes are linked together if they have some common relationships. Note that we assume the weight of edges are all one, i.e., unweighted graph in this work. Detecting the community structure of a dynamic social network is different to detecting the community structure of a static network. One of the better ways is to update the community structure by an efficient algorithm at different timestamps instead of applying a community detection procedure in every graph. For this propose, CUT traces and updates the community structure by previous known in-

Community Update and Tracking

Part1: Initial //Execute only once
Input: nodes and edges at the initial timestamp
1. Find 3-Cliques;
2. FindSeedofCommunity();
Output: seeds of communities of initial social network
Part2: Update and Track
Input: nodes and edges at the next timestamp
1. Detect variations of network;
2. While(there are some changes)
3. UpdateSOC_New();
4. UpdateSOC_Remove();
5. ExtendCommunities();
Output: communities at the current timestamp

Figure 1: Community Update and Tracking Algorithm



(a) Example social network and 3-clique (b) Example of CAB

Figure 2: Example social network and example of CAB

formation instead of recalculating the relationships between nodes and edges in the graph. We define seed of community as the core of community structure. The seed of community is the most densely connected sub-graph in a community. CUT has to keep this information at each timestamp. CUT algorithm contains two parts as Fig. 1
1. At first timestamp, finding the seeds of community as previous known information .
2. At later timestamps, tracking the seeds of community and updating them if there are some changes. Then, CUT extends the seeds of community to finally communities.

CUT detects the community structure on a social network by running once time of the part 1 of CUT at initial timestamp. Then, CUT can get the seeds of community of this social network. Finally, CUT assigns the remain nodes to become communities. When the social network changing at later timestamp, CUT just need to apply the procedure of tracking and updating part at each timestamp.

3.1 Discovery of seeds of community

FindSeedofCommunity (G,V,E)

Stack A := {}; (start with an empty stack)
1. for each vertex u, set visited[u] := false;
2. push A, v;
3. while (S is not empty) do
4. u := pop A;
5. if (not visited[u]) then
6. visited[u] := true;
7. for each unvisited neighbor w of u
8. push A, w;
9. end if
10. end while

Figure 3: Procedure of finding the seeds of community

The first step of CUT is to find 3-cliques. A clique is a strong connected sub-graph which has the greatest possible edge density. In principle, we can easily image a community is consisted by cliques. But the computation cost of detecting maximum clique is very expensive and depends on the size of clique. Therefore, we choose the 3-clique based method to find the seeds of community. There are many methods can detect all 3-cliques in the graph. We apply the backtracking algorithm to find all 3-cliques. For example, Fig. 2(a) represent a 3-clique has a unique id C_1, and C_1 contains three node N_8, N_{10}, N_{11} and three edges $E_{8,10}$, $E_{8,11}$ and $E_{10,11}$. Then, based on the clique percolation method, we build up the seeds of community from 3-cliques which correspond to complete (fully connected) sub-graphs of 3 nodes. And we define the seed of community as the Definition 1.

Definition 1. Seed of community is a collection of 3-cliques. Each 3-clique share more than one edge with at least another 3-clique in the same seed of community, denoted by $S = \{s_1, s_2, \ldots, s_n\}$*, it represents the most important sub-collection of a community.*

For easily tracking and updating seeds of community, we build up Clique Adjacent Bipartite graph, abbreviated as CAB, to represent these 3-cliques instead of original nodes and edges in the social network graph. In the original graph, a 3-cliques has three nodes and three edges. We give an serial number to represent each 3-clique in original graph. We treat the serial number of 3-clique as a node in a disjoint set of CAB. Then, we transform original edges as nodes to another disjoint set in the CAB graph. In CAB graph, each clique links to three edges. For example, Fig. 2(b) shows the CAB of three 3-cliques. There are two disjoint sets $C = \{C_1, C_2, C_3\}$ and $E_b = \{E_{b_{1,2}}, E_{b_{1,4}}, E_{b_{2,4}}, E_{b_{2,3}}, E_{b_{3,4}}, E_{b_{5,6}}, E_{b_{5,7}}, E_{b_{6,7}}\}$. A 3-clique C_1 links to three edges, $E_{b_{5,6}}$, $E_{b_{5,7}}$, and $E_{b_{6,7}}$, and 3-clique C_2 links to $E_{b_{1,2}}$, $E_{b_{1,4}}$, and $E_{b_{2,4}}$, and 3-cliques C_3 links to $E_{b_{2,4}}$, $E_{b_{2,3}}$, and $E_{b_{3,4}}$ which C_2 and C_3 share the edge $E_{b_{2,4}}$.

By definition 1, CUT finds the 3-cliques which share more than one edge. CUT joins these adjacent cliques to the same seed of community. Therefore, using CAB has two reasons as

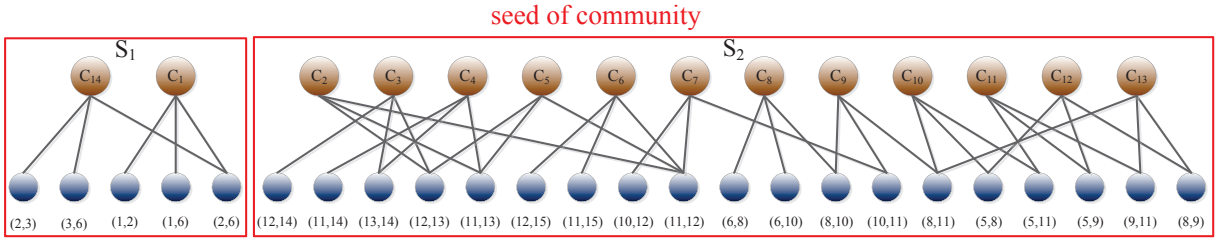


Figure 4: Seeds of community in the bipartite graph

New nodes & New edges
Input: seeds of communities(S), new nodes, new edges
Output: new seeds of communities
1. if (there are new cliques C_i) then
2. AddtoBipartiteGraph(C_i, E_{i1}, E_{i2}, E_{i3});
3. if(E_{i1}, E_{i2}, E_{i3} which any two edges link to different seeds of communities S_i, S_j) then
4. Merge(S_i, S_j);
5. else if(E_{i1}, E_{i2}, E_{i3} which any edge links to any S_i)
6. Join(S_i ,C_i);
7. end if
8. endif

Figure 5: Procedure of new nodes or edges case

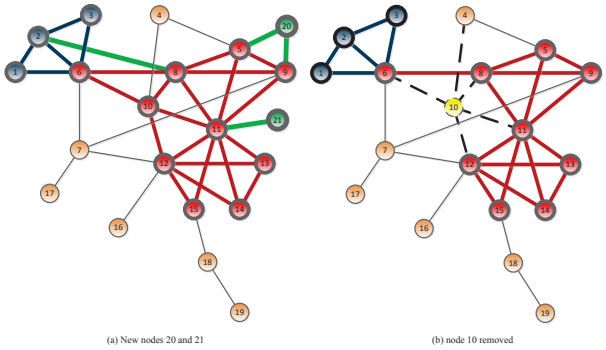


Figure 6: Some variations of social network

follows. First, two 3-cliques are connected if they share more than one edge. Therefore, traversing the connected components in bipartite graph can get the seeds of community efficiently. Second, tracing the bipartite graph and finding the connected components by depth-first-search algorithm has great complexity of computation. The connected components represent seeds of community in the bipartite graph. Fig. 3 is the procedure of tracing and finding seeds of community. Fig. 2(b) shows the seeds of community in bipartite graph, there are two seeds of community S_1 and S_2.

3.2 Tracking and updating the seeds of community

The social network always are changed with the time. There are some differences in the network between two timestamps. CUT reserve the seeds of community at each timestamp. When the new graph coming at current timestamp, CUT track the seeds of community of previous timestamp. There are four cases in updating the seeds of community, new nodes, new edges, nodes removed, and edges removed. In this section we will discuss them.

3.2.1 New edges and new nodes

Let us consider the first case. If there are new edges added into graph. CUT sweeps the adjacent edges around the new edge and combines these edges if there are new 3-cliques produced by these edges. Then, CUT adds these 3-cliques into seeds of community if they satisfy the definition of the seed of community. CUT can focus on tracking the seeds of community effectively. The seeds of community in CAB are composed by the serial number of 3-cliques and their associated 3 edges. When CUT tracking the seeds of community in CAB, CUT does not need consider the nodes in the original social network. Therefore, CUT can deal with new nodes and new edges as the same case. The procedure of new nodes and new edges is showed as the Fig. 5. There are two dynamics of seeds of community, merge and join. The join case is the edges of new clique just link to one seed of community. Another case is merge seeds of community. CUT check if the edges of new clique which any two edges link to different seeds of community S_i and S_j in the bipartite graph, then the two seeds of community are merged together. For example, as Fig. 6(a), new node 20 and 21 are added into the social network, and edges $NE = \{NE_{2,8}, NE_{5,20}, NE_{9,20}$, and $NE_{11,21}\}$ are added at the same time. CUT checks if there are new cliques produced, and add these new cliques into the bipartite graph as Fig. 7 shows. There are two new cliques C_{15} and C_{16} produced by new nodes and edges. In the bipartite graph, there is a edge $E_{b_{2,8}}$ of new clique links to S_2, we can join the clique into S_2. In addition, there are edges $E_{b_{2,6}}$, $E_{b_{6,8}}$ let S_1 and S_2 merge together.

3.2.2 Nodes removed and edges removed

The second case contains nodes removed and edges removed. No matter there are nodes or edges removed, the procedure checks if some removes lead to 3-cliques broken or not. The

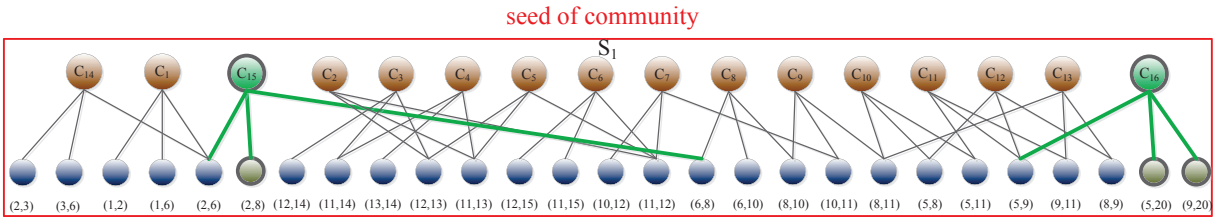


Figure 7: New clique are added into bipartite graph

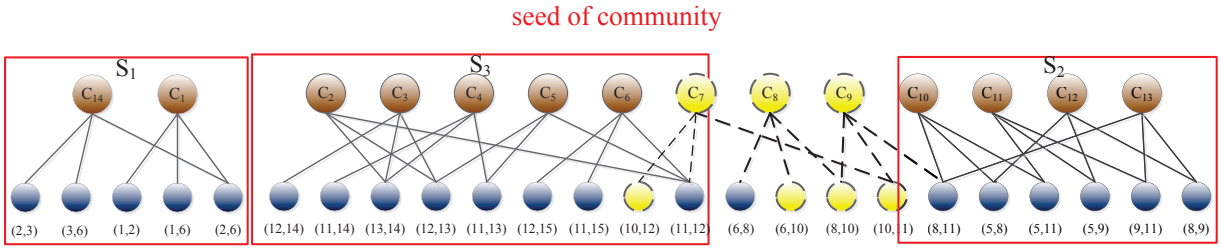


Figure 10: Cliques and edges are removed from seeds of community

procedure of remove nodes and remove edges as shown in the Fig. 8. If there are 3-cliques broken, CUT deletes the 3-cliques from the bipartite graph of seeds of community. Then, CUT updates the seeds of community by finding connected components. For example, as Fig. 6(b), the node 10 is removed from this social network. CUT finds the all edges which link to node 10 since the adjacent edges of node 10 are removed at the same time. CUT deletes these edges and cliques from the seeds of community. Executing the procedure of finding seeds of community to find the connected components again, CUT can update the seeds of community. As Fig. 10 shows the S_2 split to two small seeds of community S_2 and S_3 since there are two disjoint connected components.

Edges removed & Nodes removed
Input: seeds of communities(S), remove nodes N, remove edges E
Output: new seeds of communities
1. if(remove nodes N) then
2. E = FindAllEdge(N);
3. RemoveEdge(S,E);
4. else if(remove edges E) then
5. RemoveEdge(S,E);
6. End if

Figure 8: Procedure of nodes or edges removed case

In addition, if there are new nodes/edges and nodes/edges

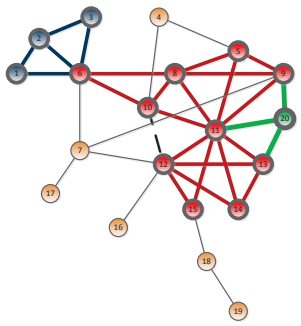


Figure 9: New node 20 added and edge(10,12) removed

removed at the same time, CUT processes the new nodes/edges case before processes the nodes/edges removed case can decrease the times of unnecessary split. The remove case may lead to split or shrunk, and the add case may lead to merge or grown. In some cases CUT can merge the seeds of community, then CUT removes some edges may not cause to split. For example, as Fig. 9, there are a new node 20 added and edge (10, 12) removed. As Fig. 11, if CUT process the new node 20 before process the edge (10, 12) removed, the new cliques C_{15} are joined into S_2 before delete C_7 from S_2 so that CUT do not need to split the seed of community and then merge them again.

3.3 Assign remain node

The seeds of communities are still not the current communities of a social network. After processing the variations of

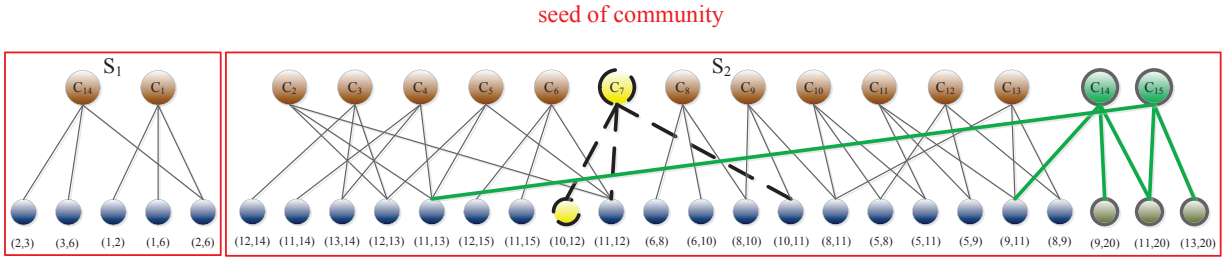


Figure 11: New seeds of community in bipartite graph

seeds of community, CUT assigns the nodes outside of seeds of community to the closet seed of community to become a community. In this procedure, CUT ignores the sparse nodes in the network first. If a node satisfies the following two conditions, we call the node is sparse node. First, the degree of node is smaller than 2. We want to ignore the bordering nodes in social network. Second, the degree of adjacency node is also smaller than α. We experimented with the α values and found that 2 gives good results across the datasets. As 9 shows, the node 19 satisfy these two conditions. These sparse nodes have less contribution to the communities. In CUT, these nodes cannot be joined into any community. CUT assigns the remaining nodes to the seeds of community which have most links to the nodes. This procedure finally can obtain the community structure in a social network.

3.4 Complexity analysis

We analyze the complexity of CUT algorithm. In initial part of CUT, finding all k-clique in an undirected graph represent a NP-complete problem. We implement the backtracking algorithm to solve this problem. The backtracking algorithm starts with each node as a clique of one. Then backtracking algorithm checks any possible merges until there are no two cliques can be merged into larger one. Each clique tests all neighbors as candidates to merge if the neighbor is connected to every node in that clique. The backtracking algorithm terminates when the size of clique equal to k. The complexity is $O(n^k)$ in which k is 3 since CUT want to find all 3-clique on a social network. Although the complexity of finding all 3-clique is expensive, but CUT just need to run the initial procedure once. At later timestamps, CUT does not apply the initial part. In the initial part, CUT also runs the procedure of find seeds of community, CUT traverse the bipartite graph to find the connected component by depth first search. The complexity is $O((|C| + |E_b|) + 3 * |C|) = O(|C| + |E_b|)$. The $|E_b|$ is the number of edges in CAB. CUT just runs the initial procedure once at initial timestamp and repeats the tracking and updating procedure when the social network changes at each later timestamp. The complexity of tracking and updating procedure, at first, let us discuss tracing procedure. CUT needs to check if there are new cliques produced, the complexity is $O(|E| * |NE|)$. The $|NE|$ is the number of new added edges in CAB. The next step, CUT checks if there are seeds of community merged or joined. The complexity which depends on the number of new cliques is $O(|C| + |E_b|)$. In the procedure of remove case, CUT has to check if there are cliques in seeds of community contain the remove edges. the complexity is $O(|C| + 3 * |C|) = O(|C|)$. Deleting the cliques which contain the removed edges from seeds of community, the complexity is same as procedure of find seeds of community.

4. EXPERIMENT

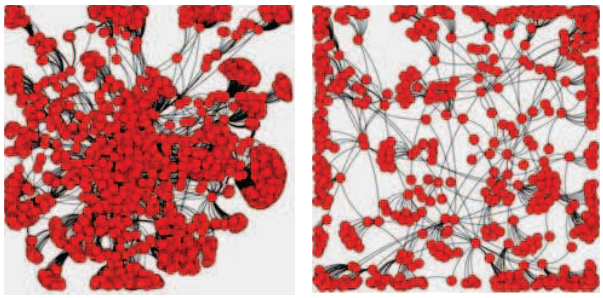
4.1 Coauthor network dataset

In this section, we test the performance of CUT algorithm on different datasets and compare with QCA algorithm. Since in QCA algorithm, their experiment result outperforms Blondel method [1] and MIEN algorithm [6], we just compare with QCA algorithm. We use modularity as a measure function to measure the overall quality of a graph partition [7]. Given a network graph and divided into several partitions, and define a matrix C where the element C_{ij} represents the total fraction of edges which stating at a node in group i and ending at a node in group j. They further define the row sums $a_i = \sum C_{ij}$ corresponds to the fraction of edges which are connected to group i. Thus, the excepted fraction of within-community edges is a_i^2. The actual fraction of edges within each group is C_{ii}. The modularity function defined by $Q = \sum_i (C_{ij} - a_i^2)$. Modularity well considered its robustness and usefulness that closely agrees with intuition on a large scale network.

In our experiment, one real dataset is the coauthor network. The nodes in this graph represent authors of article and the edges represent the co-authorship between two authors. We build the coauthor network from DBLP. DBLP is a computer science bibliography website hosted. DBLP listed more than 2.1 million articles on computer science before November 2012. Each article has title, authors, published date, conference name, journal name, book name, and website address. For satisfying the new nodes or edges cases and nodes or edges removed case, we cut the network by a time period. We capture the data of each time period from DBLP to build the coauther network. Between two time period, there are some articles added or deleted. If two authors have co-authorship in a time period, there is an edge between these two authors. In addition, if the same two authors have no co-authorship in next time period, the edge between them are removed. We take five years data as a time period. In this experiment, the first time period t_0 is 2000 to 2004, the next time period t_1 is 2001 to 2005, and so on. Fig. 12(a) shows a part of coauthor network at time period t_0. We run the initial part of CUT at the first time

period t_0. When the next network coming, CUT obtains the seeds of communities and update the community structure.

We choose several important authors of data mining area as seeds, then propagating to two levels of their coauthors. There are about 20k authors in first time period t_0. The coauthor network is densely connected graph, and the variations of network between two time periods are small. We first run the initial part of CUT to find the seeds of community and communities in the network at the first time period t_0. Note that the time period t_1 is the first time to track and update the community structure, t_2 is the second time, and so on. As Fig. 13 shows, when updating the community structure CUT has better performance on running time. The running time of CUT is almost stable at each time stamp. But CUT loses a little in modularity since CUT is not based on optimizing modularity and regards the bordering nodes as several independent communities. The modularity in the network of CUT is decreasing as Fig. 14 shows.



(a) coauthor network (b) p2p-Gnutella network

Figure 12: Coauthor network and p2p-Gnutella network

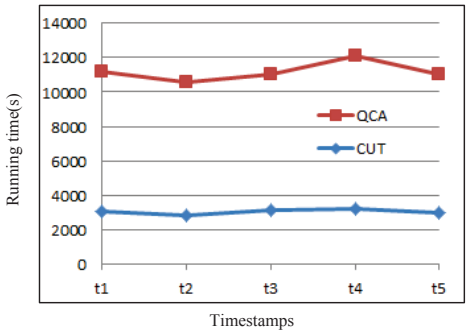


Figure 13: Running time simulation on coauthor network

4.2 p2p-Gnutella network dataset

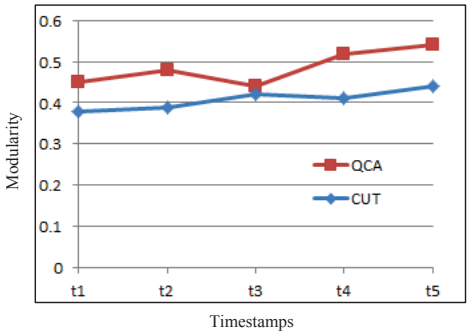


Figure 14: Modularity on coauthor network

Another social network in our experiment is p2p-Gnutella network [15]. Nodes represent hosts in the Gnutella network topology and edges represent connections between the Gnutella hosts. The network recorded the host and the connections between the hosts by a day from August 5th to August 9th, 2002. There are about 6k nodes in Gnutella network. This network is a sparse graph and changed frequently. As Fig. 12(b) shows a small part of Gnutella network. We run the initial part of CUT at the first time stamp t_0, August 5th, 2002. Then, we update the community structure by CUT at each time stamp.

In this network, the connections between the Gnutella hosts are usually changed soon. Thus, the variations of nodes and edges are large between each time stamp. As Fig. 15 shows, CUT still can deal with this type of network well on running time, but the modularity of sparse graph is lower and shown as Fig. 16.

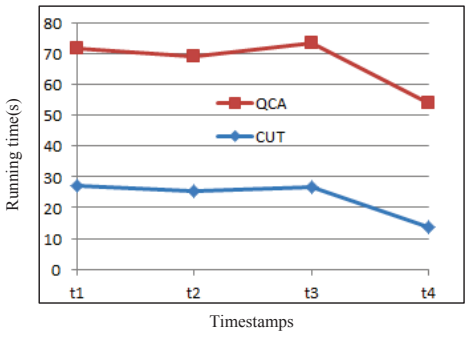


Figure 15: Running time simulation on p2p-Gnutella network

5. CONCLUSIONS

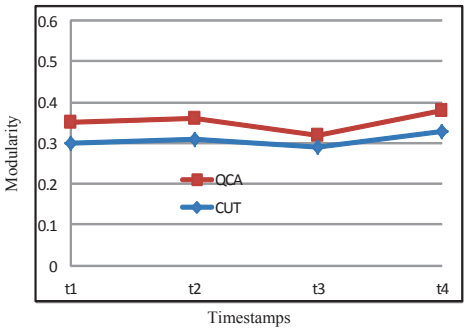


Figure 16: Modularity on p2p-Gnutella network

We propose a Community Update and Tracking (CUT) algorithm to track the seed of community in a dynamic social network, and deal with the network variations to update communities. In order to find the communities by maximizing modularity, our contribution is discovering the approximate community structure and updating the community structure efficiently in dynamic social network. CUT can update the community structure efficiently.

Considering memory space, when updating the community structure, CUT only keep seeds of communities can be better than the whole communities. Moreover, using bipartite graph to track and update the seed of community has better performance on time complexity. Experimental results show that the performance of CUT outperforms that of other methods in terms of the running time.

6. REFERENCES

[1] V. D. Blondel, J. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, 2008.

[2] R. Bourqui, F. Gilbert, P. Simonetto, F.Zaidi, U. Sharan, and F. Jourdan. Detecting structural changes and command hierarchies in dynamic social networks. *Social Network Analysis and Mining, International Conference on Advances*, pages 83–88, 2009.

[3] R. Cazabet, F. Amblard, and C. Hanachi. Detection of overlapping communities in dynamical social networks. In *Proceedings of Social Computing (SocialCom), 2010 IEEE Second International Conference*, pages 309–314, 2010.

[4] Z.-Z. Chen, K. A. Wilson, Y. Jin, W. Hendrix, and N. F. Samatova. Detecting and tracking community dynamics in evolutionary networks. In *Proceedings of Data Mining Workshops (ICDMW), 2010 IEEE International Conference*, pages 318–327, 2010.

[5] G. Clauset, M. E. J. Newman, and C. Moore. Finding community structure in very large networks. *Physical Review E*, vol. 70, 2004.

[6] T. N. Dinh, Y. Xuan, and M. T. Thai. Towards social-aware routing in dynamic communication networks. In *Performance Computing and Communications Conference (IPCCC), 2009 IEEE 28th International*, pages 161–168, 2009.

[7] M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, vol. 99:pp. 7821–7826, 2002.

[8] G.Palla, I. Derenyi, I. Farkas, and T. Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, vol. 435:pp. 814–818, 2005.

[9] S. Gregory. Finding overlapping communities in networks by label propagation. *New Journal of Physics*, vol. 12, 2010.

[10] R. Kumar, J. Novak, and A. Tomkins. Structure and evolution of online social networks. In *Proceedings of 12th ACM SIGKDD international conference on Knowledge discovery and data mining (2006)*, pages 611–617, 2006.

[11] M. E. J. Newman. Fast algorithm for detecting community structure in networks. *Physical Review E*, vol. 69, 2004.

[12] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, vol. 69, 2003.

[13] N. P. Nguyen, Y. X. T. N. Dinh, and M. T. Thai. Adaptive algorithms for detecting community structure in dynamic social networks. In *Proceedings of the IEEE Conference on Computer Communications*, pages 2282–2290, 2011.

[14] G. Palla, P. Pollner, A. Barabasi, and T. Vicsek. Social group dynamics in networks. *Adaptive Networks*, pages 11–38, 2009.

[15] M. Ripeanu, I. Foster, and A. Iamnitchi. Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *IEEE Internet Computing Journal*, vol. 6:2002, 2002.

[16] C. Tantipathananandh, T. Berger-Wolf, and D. Kempe. A framework for community identification in dynamic social networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 717–726, 2007.

[17] Y. Wang, B. Wu, and X. Pei. Commtracker: A core-based algorithm of tracking community evolution. In *Proceedings of the 4th international conference on Advanced Data Mining and Applications*, pages 229–240, 2008.

[18] B. Yan and S. Gregory. Detecting communities in networks by merging cliques. In *Proceedings of 2009 IEEE International Conference on Intelligent Computing and Intelligent Systems*, pages 832–836, 2009.

[19] S. Yang, B. Wu, H. Long, and B. Wang. Commtrend: an applied framework for community detection in large-scale social network. In *Proceedings of the 6th international conference on Fuzzy systems and knowledge discovery*, pages 139–143, 2009.