Artif Intell Rev
DOI 10.1007/s10462-011-9250-x

Incremental K-clique clustering in dynamic social networks

Dongsheng Duan · Yuhua Li · Ruixuan Li · Zhengding Lu

Abstract Clustering entities into dense parts is an important issue in social network analysis. Real social networks usually evolve over time and it remains a problem to efficiently cluster dynamic social networks. In this paper, a dynamic social network is modeled as an initial graph with an infinite change stream, called *change stream model*, which naturally eliminates the parameter setting problem of snapshot graph model. Based on the change stream model, the incremental version of a well known k-clique clustering problem is studied and incremental k-clique clustering algorithms are proposed based on local DFS (depth first search) forest updating technique. It is theoretically proved that the proposed algorithms outperform corresponding static ones and incremental spectral clustering algorithm in terms of time complexity. The practical performances of our algorithms are extensively evaluated and compared with the baseline algorithms on ENRON and DBLP datasets. Experimental results show that incremental k-clique clustering algorithms are much more efficient than corresponding static ones, and have no accumulating errors that incremental spectral clustering algorithm has and can capture the evolving details of the clusters that snapshot graph model based algorithms miss.

Keywords Incremental k-clique clustering · Dynamic social network · Change stream model

D. Duan · Y. Li (✉) · R. Li · Z. Lu
Intelligent and Distributed Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, 430074 Wuhan, People's Republic of China
e-mail: idcliyuhua@hust.edu.cn

D. Duan
e-mail: duandongsheng@smail.hust.edu.cn

R. Li
e-mail: rxli@hust.edu.cn

Z. Lu
e-mail: zdlu@hust.edu.cn

Published online: 29 May 2011

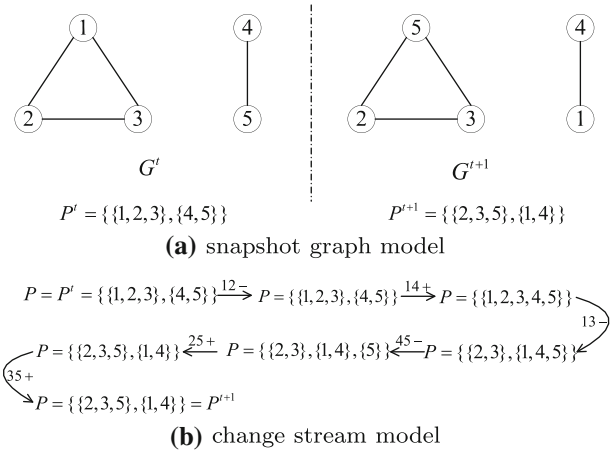


Fig. 1 The difference between evolutionary clustering and incremental clustering

1 Introduction

Clustering entities into dense parts can discover interesting groups in real or online social networks, such as amateurs with the same hobbies, friends with frequent contacts, scientists with the same research area and words with the similar semantics (Palla et al. 2005). Real social networks usually evolve over time. In the academic network, for example, new researchers join via publishing their first papers and old ones withdraw due to retirement and they might also change co-authorships during their research career. As another example, email is one of the most primary tools for people to communicate with each other and the frequency of email communications among people changes constantly. Clustering on the time-evolving social networks remains an open problem. The state-of-the-art approach of clustering in a dynamic fashion is evolutionary clustering (Chakrabarti et al. 2006). However, evolutionary clustering is a snapshot graph model based algorithms, which has some inherent drawbacks as follows.

Firstly, snapshot graph model separates the time into time-slices according to a predefined time interval. Unfortunately, it is rather difficult for users to set a reasonable time interval. On the contrary, incremental clustering tracks the dynamics of the social network in the granularity of a small change, i.e. edge or node deletion or addition, rather than a time-slice, thus eliminates the parameter setting problem of snapshot graph model.

Next, when the time interval is improperly specified, the evolving details of the clusters in the dynamic social network might be missed by snapshot graph model. Figure 1a shows the snapshot graph model, which computes a clustering result for each time-slice. However, there may be a sequence of small changes from G^t to G^{t+1}. One possible such change sequence is $\{12-, 13-, 45-, 14+, 25+, 35+\}$, where $uv-$ represents deletion of edge uv and $uv+$ represents addition of edge uv. Snapshot graph model regards the change sequence as a whole such that it loses detailed dynamics of the clusters. On the contrary, incremental clustering models the dynamics as a change stream as Fig. 1b shows, and it tracks the changes sequentially and discovers interesting evolving details of the clusters.

Lastly, although snapshot graph model is a dynamic model, it reports the clustering result for each time-slice in a static fashion. Therefore, it has to re-compute the clustering result whenever a new time-slice comes. When the change of clusters between continuous

time-slices is small, re-computation is completely unnecessary. In contrast, incremental clustering updates the clusters according to a change sequence locally thus can save lots of computational resources.

Motivated by the above comparisons, we study incremental clustering in dynamic social networks. There are two non-trivial challenges to achieve this goal. The first one is the efficiency problem, i.e. the algorithm should be conducted as fast as possible. The second one is the accuracy problem, i.e. the updated clustering result should be consistent with that produced by static one. However, the first one is the common goal of an incremental approach, and it is also the main focus of this paper. As for the second one, we study the incremental version of one kind of clustering approach which produces consistent result from different running, and K-clique clustering is a well known approach of such kind. Here, parameter k is used to constraint the density of edges of intra-clusters. The larger the k value, the denser the discovered clusters. Although setting different k values produces different clustering results, the clusters are always fixed when the k is set to some particular value.

Moreover, k-clique clustering has some other superior properties. K-clique clustering allows multiple cluster membership for nodes, which is always true for real social networks in which one entity may belong to several clusters. K-clique cluster is a strong connected part in which there are few cut-edges or cut-nodes whose removal will disjoin the cluster. K-clique clustering always produces consistent clusters when k is specified so that there is not any uncertainty in the clustering process.

Above all, incremental k-clique clustering problem in dynamic social networks is studied in this paper. The main contributions are as follows.

- A dynamic social network is modeled as an initial graph with an infinite change stream, called change stream model, which eliminates the parameter setting problem of snapshot graph model.
- Incremental k-clique clustering problem is well defined and solved based on local DFS (depth first search) forest updating. The proposed algorithms are theoretically superior to the corresponding static algorithms and incremental spectral clustering in terms of the time complexity.
- The performances of the proposed algorithms are extensively evaluated on real dynamic social networks. Experimental results show that our algorithms outperform all the compared algorithms in efficiency, and avoids the accumulating errors of incremental spectral clustering and also can capture evolving details that snapshot graph model based algorithms leave out.

The rest of the paper is organized as follows. Section 2 reviews the related works. Incremental k-clique clustering problem is formally defined in Sect. 3 and corresponding algorithms are detailedly elaborated in Sect. 4. Section 5 analyzes the time complexity of the proposed algorithms. Section 6 evaluates the efficiency and effectiveness of the proposed algorithms through experiments on real dynamic social networks and we conclude this paper in Sect. 7.

2 Related works

In this section, related works about static clustering, dynamic clustering and incremental clustering for social networks are reviewed. The works about k-clique clustering are discussed at the end of this section.

2.1 Static clustering

Clustering algorithms for static networks can be roughly categorized as measurement based and probability model based methods. The most popular measurements are normalized cut (Shi and Malik 1997) and modularity (Newman 2006). For measurement based approach, clustering problem is to search clusters that optimize corresponding evaluation measure. Due to the NP-Completeness of the measure optimization problem, heuristics (Girvan and Newman 2002) and spectral approaches (von Luxburg 2007) are proposed to obtain approximately optimal results. There are also large volume of probability model based clustering approaches (Hofmann 1999; Cohn and Chang 2000; Airoldi et al. 2008; Yang et al. 2010). Recently, clustering on heterogenous networks are studied (Sun et al. 2009). However, all the above methods deal with static networks while incremental clustering can handle dynamic ones. Most of the methods, except information theoretic clustering and k-clique clustering, require pre-specified number of clusters.

2.2 Dynamic clustering

There are also large volume of works studying dynamic social network clustering. Chakrabarti et al. (2006) put forward evolutionary clustering, in which two cost functions are presented, sq and hq, which control the snapshot quality and historical quality of clustering results respectively. Following the framework of evolutionary clustering, Chi et al. (2007); Lin et al. (2009a); Bron and Kerbosch (2009) propose novel instantiated evolutionary solutions for clustering dynamic networks. Many other works (Sun et al. 2007; Asur et al. 2007; Yang et al. 2009; Greene et al. 2010; Tantipathananandh et al. 2007; Duan et al. 2009) study the evolutionary process of clusters in dynamic social network. Recently, the community evolution problem is studied (Lin et al. 2009b; Tang et al. 2008; Sun et al. 2010) on heterogenous networks, in which there are more than one types of entities and relationships. Overall speaking, the above methods model dynamic networks as a sequence of snapshot graphs while we regard the dynamics of networks as a change stream.

2.3 Incremental network clustering

There exist research literatures on incremental clustering for network data using techniques other than k-clique clustering method. The techniques include spectral clustering (Ning et al. 2010) and minimum-cut tree based partitioning (Saha and Mitra 2006; Görke et al. 2009). These clustering techniques require pre-specified number of clusters while k-clique clustering does not. Moreover, incremental spectral clustering has computational error while our incremental k-clique clustering can be executed continuously and produces accurate clustering result along the change stream. As the experimental section shows, incremental k-clique clustering algorithms can be executed much more efficiently than incremental spectral clustering algorithm.

2.4 K-clique clustering

Comparing to hard clustering methods, k-clique clustering can detect overlapping clusters, which becomes quite popular in the last few years. For k-clique clustering problem, Palla et al. (2005) propose clique percolation method (CPM). It is based on the concept that the internal edges of a community are likely to form cliques due to their high density. Derenyi et al. (2005) study the percolation properties of k-cliques on random graphs when the edge

probability p varies. There are some works extending CPM to analyze the clustering of weighted, directed and bipartite networks (Farkas et al. 2007; Lehmann et al. 2008). Du et al. (2008) use bi-cliques as the main ingredients to detect communities in bipartite network. Palla et al. (2007) develop a CPM based algorithm to quantify the social group evolution, which allows to investigate the time dependence of overlapping communities and to uncover basic relationships characterizing community evolution. However, the method still adopts static CPM to perform k-clique clustering for each snapshot network. Overall, static k-clique clustering has been widely studied and applied in practical social networks, such as co-author network, communication network (Palla et al. 2007), word association network and molecular-biological network (Palla et al. 2005).

Static k-clique clustering, i.e. CPM, is the basis of our study. The incremental version of k-clique clustering is a very important issue for clustering dynamic networks but has been largely ignored by previous works, inspired by which we systematically study incremental k-clique clustering problem in this paper.

3 Problem definition

In this section, incremental k-clique clustering problem is defined and some necessary definitions and notations are formally presented.

Definition 1 (social network) A social network is formally defined as a graph $G(V, E)$, where node set V denotes entities in the social network and edge set $E = \{uv|u, v \in V\}$ denotes relationships between entities.

Definition 2 (k-clique) A k-clique is a complete subgraph with k nodes.

Definition 3 (k-clique cluster) A k-clique cluster is a union of all k-cliques that can be reached from each other through a series of adjacent k-cliques (two k-cliques are adjacent to each other if and only if they have k-1 common nodes).

Definition 4 (k-clique clustering) k-clique clustering is to compute all the k-clique clusters $P = \{p|p\ is\ a\ k-clique\ cluster\}$ in a graph G.

Definition 5 (dynamic social network) A dynamic social network is formally defined as an initial graph G plus an infinite change stream $c_1, \ldots, c_\infty$, where each change c_i has one of the following types, edge deletion, node deletion, edge addition and node addition, which are denoted as $uv-$, $u-$, $uv+ and u+$ respectively.

For example, the left side of Fig. 2 illustrates some kinds of k-cliques. On the right side of Fig. 2, 3-clique clusters of an artificial social network is shown. Nodes with the same color form a 3-clique cluster and black nodes belong to more than one clusters.

Based on the definitions above, incremental k-clique clustering problem in dynamic social networks is defined as Prob. 1.

Problem 1 (incremental k-clique clustering in dynamic social networks) Suppose G is the current network and P is the k-clique clusters of G. When a change c in the change stream is coming, how to locally update P such that it is the right k-clique clusters of the changed network $G' = G + c$.

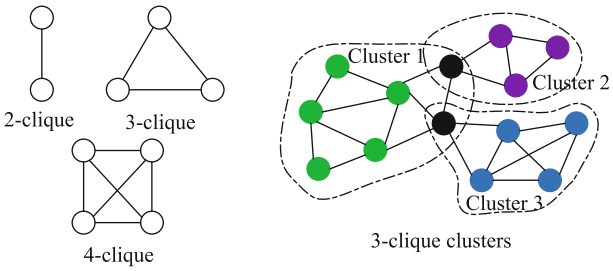


Fig. 2 K-cliques and k-clique clusters

Algorithm 1 Local DFS Forest Updating

Input: graph G, DFS forest F and change c
Output: updated G and F
1: **if** type of c is *edge deletion* **then**
2: call TED(G, F, $c.u$, $c.v$)
3: **if** type of c is *node deletion* **then**
4: **for all** neighbors u of $c.v$ in G **do**
5: call TED(G, F, u, $c.v$)
6: delete node $c.v$ from G and F
7: **if** type of c is *edge addition* **then**
8: call TEA(G, F, $c.u$, $c.v$)
9: **if** type of c is *node addition* **then**
10: add node $c.v$ into G and F
11: **return** updated G and F

4 Incremental K-clique clustering algorithms

Following the principle from specific to general, incremental 2-clique clustering is studied first and then incremental k-clique clustering ($k \geq 3$).

4.1 Incremental 2-clique clustering

Two-clique clusters are actually the connected components of a graph, which can be efficiently solved by depth first search (DFS). The clustering result can be expressed by a DFS forest F in which nodes from the same DFS tree form a cluster. Connected components can also be represented by spanning trees which can be updated by using dynamic trees and topology trees (Frederickson 1983). However, a spanning tree is not necessarily a legal DFS tree but the contrary is right. DFS tree of a graph can be applied to many other applications besides discovering connected components. In this paper, DFS tree is chosen to represent connected components.

The main part of 2-clique clustering is to obtain DFS forest F of graph G. Thus, incremental 2-clique clustering is converted to local DFS forest updating according to a given change c. The overall framework of local DFS forest updating is outlined in Algorithm 1. In this algorithm, TED(two-clique edge deletion) and TEA(two-clique edge addition) are two primary ingredients.

It is worthy to point out that there are totally two types of edges in a simple graph with respect to a DFS forest, i.e. forward edge and backward edge. Forward edges are the edges in DFS forest F while backward edges are those in the graph G but not in the DFS forest F. In addition, the two nodes adjacent to a backward edge must satisfy ancestor-descendant relation.

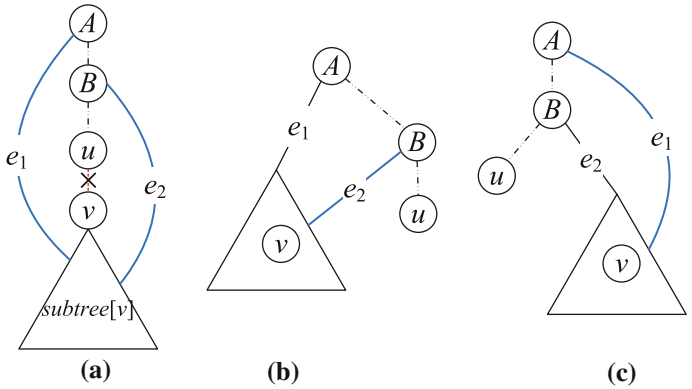


Fig. 3 An example to show which ancestor should be selected

4.1.1 Two-clique edge deletion

It is obvious that deleting a backward edge does not make any effect to DFS forest because backward edges are not in any DFS tree. Therefore, if the edge for deletion is a backward edge, we just delete it from G and keep F unchanged.

When the deleted edge is a forward edge, the situation becomes slightly complicated. The forward edge for deletion is supposed to be uv and u is the parent of v in F. Deleting uv from F makes the subtree rooted at node v(denoted as $subtree[v]$) detached from the DFS tree containing node v(denoted as $tree[v]$). However, whether $subtree[v]$ is really detached from $tree[v]$ depends on whether there exist ancestors of v connecting to $subtree[v]$ through backward edges. If there is no such ancestor, then $subtree[v]$ is truly detached from $tree[v]$, otherwise $subtree[v]$ can be re-connected to $tree[v]$ through a backward edge.

When there are many candidate ancestors of v connecting to $subtree[v]$ through backward edges, we arguably conclude that the nearest ancestor of v should be chosen, as it is the only way to make the updated DFS tree legal. Figure 3 justifies why we should choose the nearest ancestor.

Figure 3a is the original DFS tree. In this DFS tree, node v has two ancestors A and B connecting to $subtree[v]$ via backward edges e_1 and e_2 respectively. After deleting edge uv, $subtree[v]$ is detached from the DFS tree. However, $subtree[v]$ can be re-connected to the DFS tree through e_1 or e_2. If $subtree[v]$ is re-connected to node A, the adjacent nodes of edge e_2 violate ancestor-descendant relation in Fig. 3b, thus the DFS tree becomes illegal. If $subtree[v]$ is re-connected to node B, e_1 is still a legal backward edge as Fig. 3c shows. Here, B is right the nearest candidate ancestor of v. Rigorously, the nearest candidate ancestor must be selected in this case, since if the further ancestors are selected there are always illegal backward edges after the updating.

The above selected ancestor of node v is denoted as $alt[v]$, which can be formally expressed as Eq. (1),

$$alt[v] = \begin{cases} \arg \max\limits_{u \in cand[v]} order[u] & \text{if } cand[v] \neq \emptyset \\ v & \text{otherwise} \end{cases} \tag{1}$$

where $cand[v]$ is the candidate set of ancestors of v connecting to $subtree[v]$ through backward edges, $order[u]$ is the DFS order of node u. Thus, $alt[v]$ is the nearest ancestor of v

Algorithm 2 TED

Input: graph G, forest F, node u and v
Output: updated G and F
1: delete uv from G
2: **if** uv is a forward edge **then**
3: **if** $alt[v] = v$ **then**
4: detach $subtree[v]$ from $tree[v]$
5: add $subtree[v]$ to F
6: **else**
7: $S \leftarrow \{alt[v]\} \cup \{w | w \in subtree[v]\}$
8: $t \leftarrow DFS(G(S))$
9: **for all** child cld of $root(t)$ **do**
10: connect cld to $alt[v]$ as a child tree
11: **return** updated G and F

in $cand[v]$. Based on the definition of $alt[v]$, we go on describing the details of deleting a forward edge as following.

If $alt[v] = v$, then $subtree[v]$ is really detached from $tree[v]$ due to uv's deletion, because uv is the only edge or path connecting $subtree[v]$ to $tree[v]$. In this case, $subtree[v]$ is cut from $tree[v]$ and it is added to forest F as a new DFS tree. In practical applications, this kind of edge deletion separates one large cluster into two small ones.

If $alt[v] \neq v$, there are ancestors of v connected to $subtree[v]$ via backward edges. In this case, $subtree[v]$ is not truly detached from $tree[v]$ since there exist backward edges through which we can re-connect $subtree[v]$ to $tree[v]$. One of these backward edges is between $alt[v]$ and some node in $subtree[v]$. Because of the deletion of uv, the DFS order of nodes in $S = \{alt[v]\} \cup \{w | w \in subtree[v]\}$ might change. Thus, DFS must be re-performed on subgraph $G(S)$ starting from node $alt[v]$. After DFS on $G(S)$, the resultant DFS tree is connected to the original DFS tree in the position of node $alt[v]$. The pseudo code of algorithm TED is shown in Algorithm 2.

4.1.2 Two-clique edge addition

It is observed that there are totally three possible types of edges added to a graph, (1) backward edge, (2) edge between two nodes violating ancestor-descendant relation in the same DFS tree and (3) edge crossing two DFS trees.

For the addition of edge of type 1 (backward edge), it is obvious that the DFS forest F keeps unchanged.

For the case of adding an edge of type 2, the added edge is supposed to be uv as Fig. 4 shows. The nearest common ancestor of u and v is node A. Due to the addition of edge uv, the child tree of node A that contains node v (as the triangle area shows) can be directly reached from node u when conducting DFS. In this case, DFS is re-conducted on $G(triangle)$ (subgraph of G induced by node set in the triangle) starting from node v and then the resultant DFS tree is connected to node u. The new DFS order of nodes in the triangle is indicated by the red arrow. Notice that edge AC becomes a backward edge after the change thus is deleted from the tree. The right side of Fig. 4 is the updated DFS tree after adding edge uv to G.

When adding an edge of type 3, Fig. 5a shows a general case. Due to the addition of edge uv, DFS tree t ($tree[u]$) and r ($tree[v]$) is combined into a larger DFS tree. The combination operation is conducted as follows, (1) DFS on graph $G(r)$ starting from node v and (2) connect the resultant DFS tree to node u.

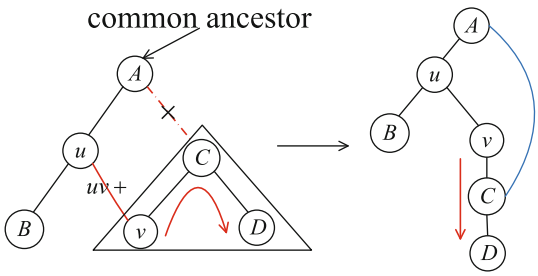


Fig. 4 Adding an edge of type 2

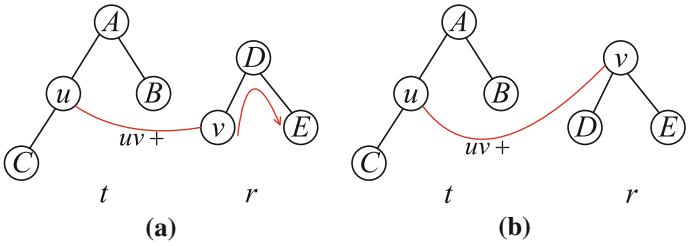


Fig. 5 Adding an edge of type 3

The new DFS orders of nodes in r are also exhibited by the red arrow. In this example tree r is connected to tree t, however tree t can also be connected to tree r because u and v are symmetric. In practical implementation, the smaller DFS tree is connected to the larger one to reduce the size of graph requiring DFS. In Fig. 5a, tree $r(size = 3)$ is smaller than tree $t(size = 4)$, so r is connected to t according to the above rule. In addition, Fig. 5b shows a special case, where node v is exactly the root of tree r. In the special case, tree r is directly connected to tree t as a child of node u and it is not necessary to conduct DFS on r. More details about the algorithm TEA is listed in Algorithm 3.

Algorithm 3 TEA

Input: graph G, forest F, nodes u and v
Output: updated G and F
add uv into G
if tree[u]=tree[v] **then**
  $w \leftarrow$nearest common ancestor of u and v
  if $w \neq u$ and $w \neq v$ **then**
    $t \leftarrow w$'s child tree that contains v
    detach t from $tree[u]$
    $R \leftarrow \{x | x \in t\}$
    DFS on $G(R)$ starting at node v
    connect the new DFS tree to node u
else
  $t \leftarrow tree[v]$
  $R \leftarrow \{x | x \in t\}$
  remove t from F
  DFS on $G(R)$ starting at node v
  connect the new DFS tree to node u
return updated G and F

Algorithm 4 Incremental K-clique Clustering

Input: G, C, H, F, P, k and c
Output: updated G, C, H, F and P
1: **if** type of c is *edge deletion* **then**
2: call KED(G, C, H, F, P, k, $c.u$, $c.v$)
3: **if** type of c is *node deletion* **then**
4: **for all** neighbor u of $c.v$ in G **do**
5: call KED(G, C, H, F, P, k, u, $c.v$)
6: delete node $c.v$ from G and F
7: **if** type of c is *edge addition* **then**
8: call KEA(G, C, H, F, P, k, $c.u$, $c.v$)
9: **if** type of c is *node addition* **then**
10: add node $c.v$ into G

4.2 General incremental K-clique clustering

In the previous subsection, local DFS forest updating algorithm is proposed for incremental 2-clique clustering problem. However, 2-clique clustering is the most special case of k-clique clustering. The general incremental k-clique clustering ($k \geq 3$) is discussed in this subsection.

As a preliminary of incremental k-clique clustering, the static k-clique clustering algorithm (Everett and Borgatti 1998) is reviewed first. The procedures of k-clique clustering are as follows. (1) Find all the maximal cliques $C = \{C_1, \ldots, C_n\}$ with size $\geq k$ in G. (2) Denote each discovered clique by a clique node and connect two cliques through an edge if they have at least $k-1$ common nodes in G. These clique nodes and edges constitute a clique graph H. (3) Perform DFS on H to discover all the connected components of H in terms of the DFS forest F. According to the DFS forest F of clique graph H and the discovered cliques C, the clustering result P can be obtained trivially.

Unfortunately, maximal cliques discovery is a well known NP-Complete problem (Abello et al. 2002). There are extensive works studying maximal cliques discovery problem which is beyond the scope of this paper. We just adopt the algorithm developed by Bron and Kerbosch (1973), which is one of the fastest algorithms. Based on the static k-clique clustering, the overall framework of incremental k-clique clustering algorithm is outlined in Algorithm 4.

It can be seen that the clustering result P depends on DFS forest F and maximal clique set C, thus the main task of incremental k-clique clustering is to update F and C incrementally according to a given change c. In order to update F, the change c is mapped to a sequence of changes of the clique graph H referred to *change mapping technique* and then local DFS forest updating algorithm (Algorithm 1) is conducted on H. In the following, we introduce the change mapping details from incremental k-clique clustering to incremental 2-clique clustering.

4.2.1 K-clique edge deletion

If edge uv is deleted from G, C and H change as follows. For each clique $C_i \supseteq \{u, v\}$ with size $> k$ in C, C_i is divided into two cliques $C_i - \{v\}$ and $C_i - \{u\}$. Then, $C_i - \{u\}$ is inserted into C as a new maximal clique and C_i is replaced with $C_i - \{v\}$. The changes to clique graph H are as follows. Firstly, some existing edges of clique node i are removed if the intersection between C_i and those neighboring cliques is less than $k-1$ due to the deletion of node v from C_i. Secondly, a new clique node $n+1$ corresponding to $C_i - \{u\}$ is added to H. Thirdly, edges are added between clique node $n+1$ and other clique nodes if C_{n+1} has at least $k-1$ common nodes with those cliques.

Let's see some special cases for deleting edge uv. If there is not any clique in C containing both u and v, C and H keeps unchanged. If there is a clique C_i with size $= k$ containing both u and v, C_i is deleted from C and clique node i is removed from H. Based on the analysis above, the algorithm KED is listed in Algorithm 5.

Algorithm 5 KED

Input: $G,\ C,\ H,\ F,\ P,\ k\ ,\ u$ and v
Output: updated $G,\ C,\ H,\ F,\ P$
for all C_i in C containing u and v **do**
  if $size(C_i) > k$ **then**
    $n \leftarrow n + 1$
    $C_n \leftarrow C_i - \{u\}$
    $C_i \leftarrow C_i - \{v\}$
    add node n into H and F
    for all neighbor j of i in H **do**
      if $\left|C_i \cap C_j\right| < k - 1$ **then**
        call TED($H,\ F,\ i,\ j$)
      if $\left|C_n \cap C_j\right| \geq k - 1$ **then**
        call TEA($H,\ F,\ n,\ j$)
    call TEA($H,\ F,\ i,\ n$)
  else
    delete C_i from C
    delete i from H and F
update P according to C and F
return updated $G,\ C,\ H,\ F,\ P$

Algorithm 6 KEA

Input: $G,\ C,\ H,\ F,\ P,\ k,\ u$ and v
Output: updated $G,\ C,\ H,\ F,\ P$
$CN \leftarrow$common neighbors of u and v
$C' \leftarrow$the maximal cliques in $G(CN)$
for all C'_i in C' **do**
  $C'_i \leftarrow C'_i + \{u, v\}$
for all C'_i in C' **do**
  for all C_j in C **do**
    if $C'_i \supseteq C_j$ **then**
      delete C_j from C
      for all neighbor l of j in H **do**
        call TED($H,\ F,\ l,\ j$)
      delete node j from H and F
  $n \leftarrow n + 1, C_n \leftarrow C'_i$
  for all C_j in C **do**
    if $\left|C_n \bigcap C_j\right| \geq k - 1$ **then**
      call TEA($H,\ F,\ n\ ,\ j$)
update P according to C and F
return updated $G,\ C,\ H,\ F,\ P$

4.2.2 K-clique edge addition

While adding edge uv to G, new maximal cliques can be found in the following way. (1) Get the common neighbors of u and v in G, $CN(u, v) = \{w | w \in N(u)\ \wedge\ w \in N(v)\}$ where $N(u)$ denotes the neighbors of u. (2) Find all the maximal cliques C' (of size $\geq k - 2$) in

$G(CN(u, v))$ which is a subgraph of G induced by node set $CN(u, v)$. (3) For each clique C'_i in C', add u and v into C'_i. Then C' is the new generated maximal clique set (of size $\geq k$) after edge uv is added.

It must be noted that cliques in C' may contain some original cliques in C as a subset, so the inclusion relation between two cliques with one from C' and another from C must be tested. If C'_i in C' contains C_j in C, C_j is replaced with C'_i. If C'_i does not contain any clique in C, insert C'_i into C as a new clique and then a new clique node and some necessary edges are added into H. Algorithm 6 shows the algorithm KEA in a more detail.

5 Algorithm complexity analysis

The time complexity of the proposed algorithms is analyzed and compared with corresponding static algorithms and incremental spectral clustering algorithm in this section. Local DFS forest updating is analyzed first, and then incremental k-clique clustering and the complexity comparison comes finally.

5.1 Complexity of local DFS forest updating

The time complexity of local DFS forest updating algorithm depends on the type of the input change c. As discussed in the previous section, the two non-trivial cases are two-clique edge deletion and two-clique edge addition.

5.1.1 Two-clique edge deletion

If a backward edge is deleted, the time complexity is $O(1)$. If the deleted edge uv is a forward edge, $alt[v]$ is first computed in $O(h_v)$ time, where h_v is the height of node v in DFS tree which is usually in log scale of the total number of nodes, i.e. $h_v = O(logN)$. If $alt[v] = v$ the time complexity of subsequent processing is $O(1)$, Otherwise, it is $O(|subtree[v]|)$ which is the complexity of DFS on subgraph of G induced by nodes in $subtree[v]$. Overall, The best time complexity of deleting a forward edge is $O(logN)$ and the worst one is $O(logN + |subtree[v]|)$.

5.1.2 Two-clique edge addition

The best time complexity of edge addition is $O(1)$ (e.g. when a backward edge is added) and the worst case is $O(|tree[v]|)$ which is the complexity of DFS on subgraph of G induced by nodes in $tree[v]$.

Above all, the best time complexity of local DFS forest updating is $O(1)$, while the worst one is $O(max(logN + |subtree[v]|, |tree[v]|))$.

5.2 Complexity of general incremental K-clique clustering

The time complexity of incremental k-clique clustering algorithm depends on the type of the given change c. As discussed in the previous section, the primary cases are k-clique edge deletion and k-clique edge addition.

Table 1 Comparison of algorithms' time complexity

Algorithms	Best	Worst
Incremental K-Clique		
$K = 2$	$O(1)$	$O(max(logN + \|subtree[v]\|, \|tree[v]\|))$
$K \geq 3$	$O(1)$	$O(max(logL + \|subtree\|, \|CN\|3^{\|CN\|/3} + \|tree\|))$
Static K-Clique		
$K = 2$	$O(N)$	$O(N)$
$K \geq 3$	$O(N)$	$O(N3^{N/3})$
Incremental spectral		
$O(N)$	$O(N)$	

5.2.1 K-clique edge deletion

Let n be the number of cliques of size $> k$ including both u and v. Let d be the average degree of nodes in clique graph H. Then, the overall average time complexity of KED is $O(ndx)$, where x represents the complexity of local DFS forest updating on clique graph H. Since n and d can be regarded as constants, the time complexity of KED equals to $O(x)$. Therefore, the best time complexity is $O(1)$ and the worst one is $O(max(logL + |subtree|, |tree|)$, where $|subtree|$ and $|tree|$ denote the average number of nodes in subtrees and trees of H respectively and L is the number of maximal cliques in G.

5.2.2 K-clique edge addition

The worst time complexity of maximal cliques discovery is $O(N3^{N/3})$ (Etsuji Tomita and Tanaka 2004). Analogically, the time complexity of maximal cliques discovery on a subgraph induced by common neighbors CN of u and v (line 2 of Algorithm 6) is $O(|CN|3^{|CN|/3})$. The time complexity of the rest part equals to that of edge addition on graph H with best complexity $O(1)$ and worst $O(|tree|)$. Therefore, the total time complexity is $O(|CN|3^{|CN|/3})$ for the best case and $O(|CN|3^{|CN|/3} + |tree|)$ for the worst one.

Above all, the best time complexity of incremental K-clique clustering is $O(1)$ and the worst is $O(max(logL + |subtree|, |CN|3^{|CN|/3} + |tree|))$.

5.3 Complexity comparison

The time complexity of static 2-clique clustering is $O(N)$. The worst time complexity of static k-clique clustering is $O(N3^{N/3} + L)$ and the best one is $O(N)$. As reported in Ning et al. (2010), the time complexity (best or worst one) of incremental spectral clustering algorithm is $O(N)$.

The comparison of the best and worst time complexity of the proposed algorithms, static algorithms and incremental spectral clustering algorithm are listed in Table 1, which shows that the time complexity of incremental k-clique clustering algorithms outperforms static ones and incremental spectral clustering algorithm. Note that the average values of $|subtree[v]|$ or $|subtree|$ and $|tree[v]|$ or $|tree|$ and $|CN|$ can be regarded as constants in real social networks.

6 Experiments

In this section, incremental k-clique clustering algorithms are evaluated through experiments on two real dynamic social network datasets. Before going to the experiments, the datasets are first described.

6.1 Datasets

6.1.1 ENRON

This dataset consists of the e-mail communication data[1] of ENRON company from Sept. 27th 1999 to Mar. 5th 2000. There are totally 6,535 distinct email addresses (nodes) and 48,808 messages (edges) in the dataset. One week is chosen as a time-slice, then there are 23 time-slices in all. Summing up all the time-slices, there are totally 28,160 changes. The communicating network of the first time-slice is the initial graph and all the 28,160 changes form the change stream.

6.1.2 DBLP

This dataset consists of papers published in five proceedings (KDD, ICDM, CIKM, WWW and SIGIR) from 2006 to 2009 extracted form DBLP.[2] In this dataset, one year is regarded as a time-slice. For each year, a co-author network can be constructed by regarding authors as nodes and co-authorships as edges. To make the co-author networks being connected, just the largest connected components are preserved. After the preprocessing, there are 1,173 authors and 4,089 co-authorships left. There are totally 5,514 nontrivial changes between continuous time-slices. The co-author network in 2006 is regarded as the initial graph and all the 5,514 changes from the change stream.

6.2 Efficiency study

In this subsection, the efficiency of our proposed algorithms are studied and compared with corresponding static algorithms and incremental spectral clustering algorithm (Ning et al. 2010) through experiments on the ENRON and DBLP datasets described in the previous subsection. All the algorithms are implemented in standard C++ and conducted on PC with Core Duo 2.26 GH CPU and 3 GB RAM.

The time costs of the algorithms on ENRON and DBLP data are illustrated in Figs. 6 and 7. In the two figures, the total time cost of each time slice instead of that of each change is recorded for the purpose of comparison convenience. Since incremental and static algorithms have the same time cost in the first time-slice, thus they are omitted in the figures. For the sake of clarity, the time costs are shown in the log-scale. The comparison results show that the time costs of incremental k-clique ($k = 2, 3, 4$) clustering algorithms are much lower than those of corresponding static k-clique clustering algorithms and incremental spectral clustering algorithm, which validates the theoretic time complexity comparison results.

We also conduct experiments on k values from 2 to 9. In these experiments, the total time of all the time-slices are recorded. Figure 8 shows the results. Notice that Y-axis is shown in log-scale. From the result, it also can be observed that the incremental algorithms are

[1] http://www.cs.cmu.edu/~enron/.

[2] http://dblpvis.uni-trier.de/.

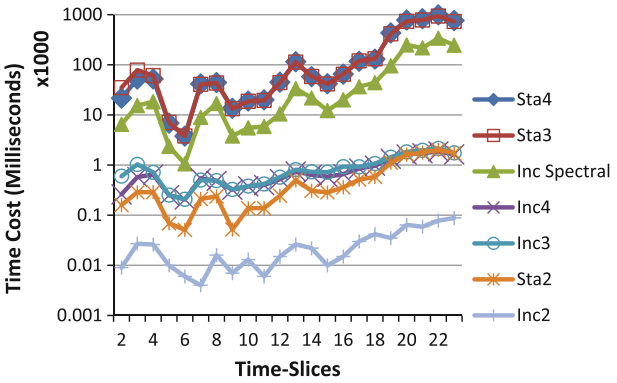


Fig. 6 The time costs of the algorithms on ENRON data

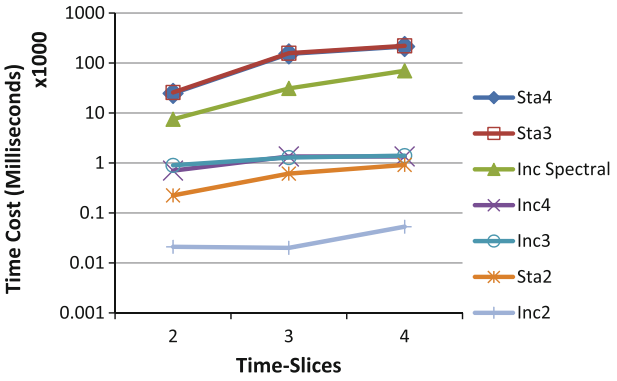
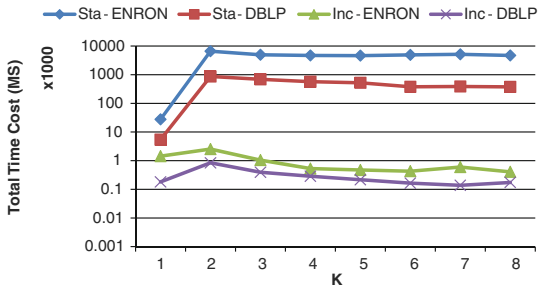


Fig. 7 The time costs of the algorithms on DBLP data

Fig. 8 Total time cost of the algorithms



executed much more quickly than static algorithms. As k increases from 3, the time costs of either incremental or static k-clique clustering algorithms decrease. That is because the larger the k value, the sparser the clique graph.

6.3 Effectiveness study

In this subsection, we study the clusters in DBLP dataset discovered by our algorithms. Only DBLP dataset is chosen for this experiment since the profiles of authors can be easily obtained from their homepages. Particularly, the incremental 3-clique clustering algorithm

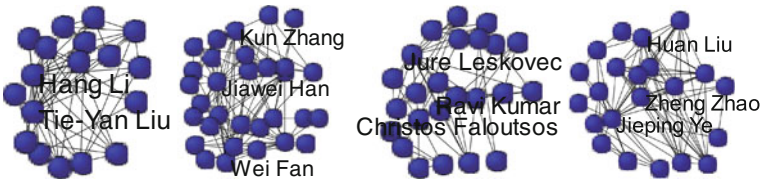


Fig. 9 Discovered clusters in DBLP dataset at the end of 2008

is applied in this subsection. The algorithm discovers several main research groups with size significantly large. These research groups are relatively stable from year 2006 to 2009, i.e. the membership are sustained during this period. However, there are new authors joining and some old authors leaving these groups and some groups are merged to a large one and large groups are divided to small ones as the network evolves. The incremental algorithm can capture these detailed evolutions.

We validate the discovered clusters through carefully investigating the lead authors' affiliations from their homepages. However, identifying a leader in a link based cluster is a well known ranking problem, for which there are large volume of measurements. Here, we just use the node's internal degree[3] to evaluate the importance of a node in a cluster. The larger the internal degree, the more likely the node is a leader.

Figure 9 shows four main clusters discovered at the end of 2008 marked by some lead members. Through carefully checking the homepages of the authors, Figure 9a corresponds to Information Retrieval and Mining Group of Microsoft Research Asia, where Tie-yan Li and Hang Li are the lead researcher and senior researcher respectively. The lead authors in Fig. 9b are from University of Illinois at Urbana-Champaign (Jiawei Han) and IBM T.J.Watson (Fang Wei), which suggests there exist collaboration between the two organizations. The leaders in Fig. 9c are from Yahoo! research and Carnegie Mellon University, which implies that the two research groups have some collaborations. Ravi Kumar is an active researcher in Yahoo! research and Christos Faloutsos is a professor of Carnegie Mellon University, who supervised Jure Leskovec. Figure 9d is a data mining research group from Arizona State University, where Huan Liu is a professor and Jianping Ye is an associate professor and Zheng Zhao is co-supervised by the above two authors.

Given a change, i.e. adding or deleting an edge, experiments show that the clusters of the DBLP network just changes locally. There are many changes not affecting any cluster. Our algorithms discover those changes which lead to significant changes to some clusters. Among 5,514 changes of DBLP network, it is found that there are 1,606 changes affecting the clusters and 125 of them directly lead to the dividing or merging of clusters while the left changes just drawing a new member to or excluding an old member from a cluster.

For example, Table 2 lists several changes which lead to the dividing or merging of some clusters. Through checking the profiles of the authors, the meanings of clusters divided or merged are given out manually.

Static k-clique clustering algorithms can also obtain the changes of the clusters by recomputing on the updated networks, but the incremental k-clique clustering algorithms are much more efficient as the previous subsection shows. Comparing to the incremental spectral clustering algorithm which gives out an approximation of updated eigenvectors leading to

[3] The number of neighboring nodes in the same cluster.

Table 2 Changes leading to dividing and merging of clusters in DBLP dataset

Small changes	Changes to the clusters
(A. Z. Broder, M. Sayyadian)−	A large Yahoo! research group is divide
(S. Vadrevu, X. Li)+	Two small Yahoo! research groups are merged
(S. Zheng, C. L. Giles)+	Two groups from PSU[a] and MSRA[b] are merged together
(W.-C. Lee, C. L. Giles)−	A large group from PSU and MSRA is divided

[a] Pennsylvania State University
[b] Microsoft Research Asia

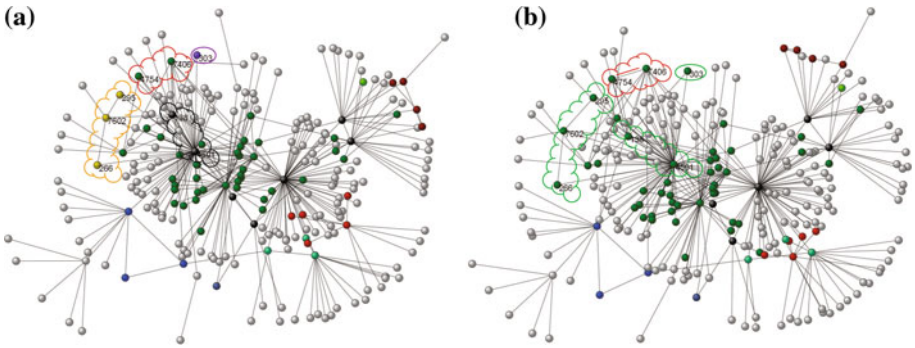


Fig. 10 The clusters **a** before and **b** after adding edge (4754, 7406)

accumulating errors, incremental k-clique clustering algorithms produce the same results as static algorithms accurately with no accumulating error.

6.4 Incremental K-clique clustering algorithm vs. snapshot graph model based algorithms

Snapshot graph model based algorithms track the clusters in the granularity of a time-slice. However, the interval of a time-slice may be arbitrarily chosen by a user, such as a day, a week or a month. Alternatively, incremental k-clique clustering algorithm tracks the clusters in the granularity of a change rather than a time-slice, thus the time interval setting problem of snapshot graph model is naturally eliminated. Moreover, incremental k-clique clustering can capture evolving details that snapshot graph model misses. The following cases justify this argument.

Figure 10 shows the clustering results[4] before and after adding edge (4754, 7406)[5]. Because of the addition of edge (4754, 7406), yellow nodes {266, 295, 7602} and purple node 303 become members of the green cluster and black nodes {138, 4501} change its color to green as they belong merely to the green cluster after the edge's addition. Figure 11 shows the clustering results before and after deleting edge (4785, 271). Due to the deletion of edge (4785, 271), the color of node 271 changes from green to white because it leaves the green cluster and nodes {4771, 7526} are separated from the green cluster and form the purple cluster.

[4] White nodes are not in any 3-clique cluster and black nodes are in more than one clusters and nodes with other colors represent independent clusters.

[5] Numbers denote the identities of nodes.

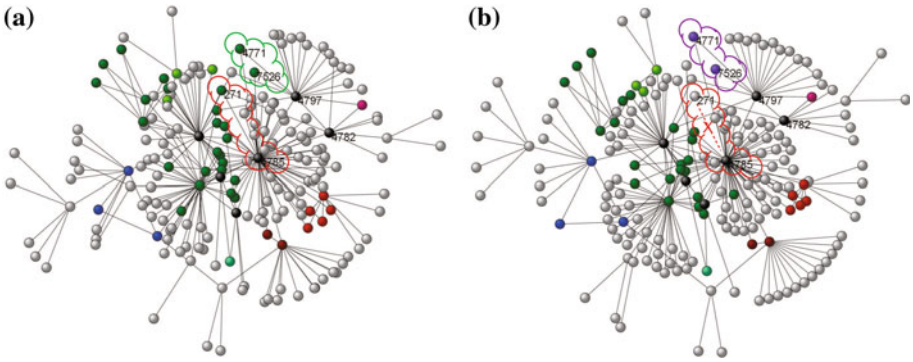


Fig. 11 The clusters **a** before and **b** after deleting edge (4785, 271)

As the above cases show, incremental k-clique clustering tracks the evolution of clusters with respect to a small change. During the 36th time-slice, experiments show that there are totally 62 intermediate changes of the clusters. Arguably, any snapshot graph model based algorithm can not find these evolving details. On the other hand, if the time interval is set to a rather short one (extremely, once a change happens a new time-slice starts), the snapshot graph model based algorithms will cost much more time than incremental algorithms since it recomputes clusters on the whole network whenever a change comes.

7 Conclusion and future work

This paper models the dynamics of social networks as a change stream. Based on this model, local DFS forest updating algorithm is proposed for incremental 2-clique clustering and then it is generalized to incremental k-clique clustering. The incremental strategies are well designed to guarantee the accuracy of the clustering result with respect to any kind of changes. Experimental results on ENRON and DBLP datasets show that the proposed algorithms are much more efficient than corresponding static algorithms and incremental spectral clustering algorithm. In addition, incremental k-clique clustering algorithms naturally avoid the accumulating error that incremental spectral clustering algorithm has. In contrast to snapshot graph model based algorithms, our algorithms can capture much more detailed changes of clusters in the granularity of a small change. Furthermore, local DFS forest updating algorithm not only gives out the updated connected components of a graph but also the updated DFS forest, which can be applied to other issues, such as finding a simple loop through a node. There are still some interesting works left for future research, including exploiting the power of the discovered changes uncovering other interesting knowledge, performing experiments on other kinds of datasets to further validate the capability of our algorithms and incorporating the textual information into the incremental k-clique clustering to track both the topical and structural changes of clusters.

Acknowledgments This work is supported by National Natural Science Foundation of China under Grant 70771043, 60873225, 60773191, National High Technology Research and Development Program of China under Grant 2007AA01Z403, Natural Science Foundation of Hubei Province under Grant 2009CDB298, Wuhan Youth Science and Technology Chenguang Program under Grant 200950431171, Open Foundation of State Key Laboratory of Software Engineering under Grant SKLSE20080718, and Innovation Fund of Huazhong University of Science and Technology under Grant Q2009021.

References

Abello J, Resende MGC, Sudarsky S (2002) Massive quasi-clique detection. In: LATIN, pp 598–612

Airoldi EM, Blei DM, Fienberg SE, Xing EP (2008) Mixed membership stochastic blockmodels. J Mach Learn Res 9(6):1981–2014

Asur S, Parthasarathy S, Ucar D (2007) An event-based framework for characterizing the evolutionary behavior of interaction. In: KDD, pp 913–921

Bron C, Kerbosch J (1973) Finding all cliques of an undirected graph. Commun ACM 16(9):575–577

Bron C, Kerbosch J (2009) A particle-and-density based evolutionary clustering method for dynamic networks. PVLDB 2(1):622–633

Chakrabarti D, Kumar R, Tomkins A (2006) Evolutionary clustering. In: KDD, pp 554–560

Chi Y, Song X, Zhou D, Hino K, Tseng B (2007) Evolutionary spectral clustering by incorporating temporal smoothness. In: KDD, pp 153–162

Cohn D, Chang H (2000) Learning to probabilistically identify authoritative documents. In: ICML, pp 167–174

Derenyi I, Palla G, Vicsek T (2005) Clique percolation in random networks. Phys Rev Lett 94(16):160–202

Du N, Wang B, Wu B, Wang Y (2008) Overlapping community detection in bipartite networks. In: WIC, pp 176–179

Duan D, Li Y, Jin Y, Lu Z (2009) Community mining on dynamic weighted directed graphs. In: CNIKM, pp 11–18

Etsuji Tomita A, Tanaka HT (2004) The worst-case time complexity for generating all maximal cliques and computational experiments. Theor Comput Sci 363:28–42

Everett MG, Borgatti SP (1998) Analyzing clique overlap. Connections 21(1):49–61

Farkas IJ, Abel D, Palla G, Vicsek T (2007) Weighted network modules. New J Phys 9(6):180

Frederickson GN (1983) Data structures for online updating of minimum spanning trees. In: SOTC, pp 252–257

Girvan M, Newman M (2002) Community structure in social and biological networks. PNAS 99(12):7821–7826

Görke R, Hartmann T, Wagner D (2009) Dynamic graph clustering using minimum-cut trees. In: WADS, pp 339–350

Greene D, Doyle D, Cunningham P (2010) Tracking the evolution of communities in dynamic social networks. In: ASONAM, pp 176–183

Hofmann T (1999) Probabilistic latent semantic analysis. In: UAI, pp 289–296

Lehmann S, Schwartz M, Hansen LK (2008) Biclique communities. Phys Rev E 78(1):016–108

Lin YR, Chi Y, Zhu S, Sundaram H, Tseng BL (2009) Analyzing communities and their evolutions in dynamic social networks. TKDD 3(2):1–31

Lin YR, Sun J, Castro P, Konuru R, Sundaram H, Kelliher A (2009) Metafac: community discovery via relational hypergraph factorization. In: KDD, pp 527–536

Newman MEJ (2006) Modularity and community structure in networks. In: PNAS, pp 8577–8582

Ning H, Xu W, Chi Y, Gong Y, Huang T (2010) Incremental spectral clustering by efficiently updating the eigen-system. Pattern Recogn 43(1):113–127

Palla G, Derenyi I, Farkas I, Vicsek T (2005) Uncovering the overlapping community structure of complex networks in nature and society. Nature 435(7043):814–818

Palla G, Barabasi A, Vicsek T (2007) Quantifying social group evolution. Nature 446(7136):664–667

Saha B, Mitra P (2006) Dynamic algorithm for graph clustering using minimum cut tree. In: ICDM workshops, pp 667–671

Shi J, Malik J (1997) Normalized cuts and image segmentation. In: CVPR'97, pp 731–737

Sun J, Papadimitriou S, Yu P, Faloutsos C (2007) Graphscope: Parameter-free mining of large time-evolving graphs. In: KDD, pp 687–696

Sun Y, Yu Y, Han J (2009) Ranking-based clustering of heterogeneous information networks with star network schema. In: KDD, pp 797–806

Sun Y, Tang J, Han J, Gupta M, Zhao B (2010) Community evolution detection in dynamic heterogeneous information networks. In: MLG-KDD, pp 137–146

Tang L, Liu H, Zhang J, Nazeri Z (2008) Community evolution in dynamic multi-mode networks. In: KDD, pp 677–685

Tantipathananandh C, Berger-Wolf T, Kempe D (2007) A framework for community identification in dynamic social networks. In: KDD, pp 717–726

von Luxburg U (2007) A tutorial on spectral clustering. Stat Comput 17(4):395–416

Yang T, Chi Y, Zhu S, Gong Y, Jin R (2009) A bayesian approach toward finding communities and their evolutions in dynamic social networks. In: SDM, pp 990–1001

Yang T, Chi Y, Zhu S, Gong Y, Jin R (2010) Directed network community detection: a popularity and productivity link model. In: SDM, pp 742–753