International Journal of Database Theory and Application
Vol.7, No.3 (2014), pp.201-218
http://dx.doi.org/10.14257/ijdta.2014.7.3.18

The Hierarchical Structure and Bridging Member of k-Clique Community [1]

Kaikuo Xu[1], Changan Yuan[2] and Xuzhong Wei[3]

[1]*College of Computer Science, Chengdu University of Information Technology, ChengDu, 610225, China*
[2]*Guangxi Teachers Education University, Nanning 530001, Chinas*
[3]*School of Computer Science, Sichuan University*
[1]*kaikuoxu@gmail.com*, [2]*yca@gxtc.edu.cn*

Abstract

Community detection is widely applied in many fields and k-clique community detection is one important detection method. There are many works on k-clique community detection. However, the work on analyzing the structure of k-clique community is rare. In this paper, we first give the definition of k-clique community tree and closed l-s-clique community, which could be used as the index of analyzing k-clique community. Then we give the definition of l-s-clique community pivot to describe the members playing the bridging roles in k-clique community. We analyze the properties of l-s-clique community and propose KCliqueTree algorithm based on the properties. This algorithm could efficiently generate k-clique community tree whose leaf nodes represent closed l-s-clique community. We also propose LSBridge algorithm to search l-s-clique community pivot. At last, we conduct case study on DBLP (Digital Bibliography & Library Project) dataset, which shows the availability of our definitions and algorithms.

Keywords: *k-Clique Community Tree, l-s-clique community, k-Clique Community, Community Detection*

1. Introduction

Graph is an abstract of real world data: the nodes indicate the information of entities while the edges indicate the links between entities [1]. To store the detailed information of the link, the edges could be abstracted as weighted directed or weighted undirected or unweighted directed or unweighted undirected. For example, in the research of scientific bibliography like DBLP [2], the authors are abstracted as nodes while the co-authorships between them are abstracted as edges. The time authors co-authored could be abstracted as the weight of the edges; the citation relationship between authors could be abstracted as edges.

Many concepts are proposed to study the properties of graph data [3, 4]. Among them, community is of great interest to the research community and many algorithms are proposed for community detection [5-9]. Since both the assumption and the algorithms are relatively simple for unweighted undirected graph comparing with those for weighted and directed graph, the results on unweighted undirected graph is the basis for further analysis on the other

[1] This work was supported the Natural Science Key Foundation of Guangxi under Grant No.2011GXNSFD018025, the Development Foundation of CUIT under grant No. KYTZ201108, the Natural Science Foundation under Grant No. 61363037 and Haikou city key science and technology program (2012-027).
[2] Corresponding author.

ISSN: 2005-4270 IJDTA

two types of graphs [10, 11]. *K*-clique community detection is for unweighted undirected graph. There is extended work on *k*-clique community detection [12-17]. However, the work on analyzing the structure of *k*-clique community is rare. Let us look at the definition first.
Definition 0: A *k-clique community* is a union of all *k*-cliques (complete subgraphs of size *k*) that can be reached from each other through a series of adjacent *k*-cliques (where adjacency means sharing *k*-1 nodes).

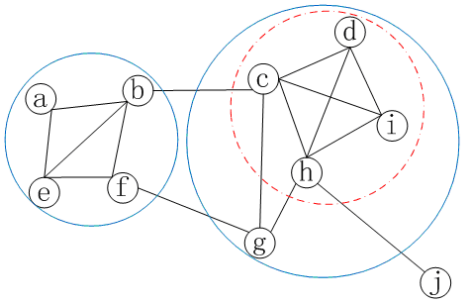


Figure 1. *K*-clique Communities with Different *k*

Example 1: Assume a tiny subset of DBLP is abstracted as the graph (unweighted undirected) in Figure 1. When *k* is set to be 2, all the vertices are in a 2-clique community. When *k* is set to be 3, *a*, *b*, *e* and *f* are in a 3-clique community while *c*, *d*, *g*, *h* and *i* are in another 3-clique community. When *k* is set to be 4, *c*, *d*, *h* and *i* are in a 4-clique community. In Fig.1, the 2-clique community could be viewed as the first level community in describing the strength of the vertices connections; the two 3-clique communities could be viewed as the second level communities and the 4-clique community could be viewed as the third level community. They together form a hierarchical structure. Furthermore, through *bc* and *fg*, {*a*, *b*, *e*, *f*} and {*c*, *d*, *g*, *h*, *i*} are in the same *2*-clique community. Then *b*, *f* in {*a*, *b*, *e*, *f*} and *f*, *g* in {*c*, *d*, *g*, *h*, *i*} could be viewed as playing the role of bridging different communities.

The hierarchical clustering method [18] was proposed to detect communities with hierarchical tree and was widely applied since then [19-21]. It requires manually definition on the distance between vertices and the hierarchical tree generated by the algorithms are for the graph instead of *k*-clique communities. There is also extended work on assessing a vertice's involvement in the walk structure of a graph [22-24]. Its focus is not on the *k*-clique communities either.

This paper focuses on *k*-clique community. The main contributions include (a) giving the definition of *k*-clique community tree and closed *l*-*s*-clique community as the index of analyzing the hierarchical structure of *k*-clique community; (b) giving the definition of *l*-*s*-clique community pivot to describe the members bridging *k*-clique communities; (c) proposing KCliqueTree algorithm to efficiently generate *k*-clique community tree whose leaf nodes represent closed *l*-*s*-clique community; proposing LSBridge algorithm to search *l*-*s*-clique community pivot. (d) conducting case study on DBLP datasets.

The rest of the paper is organized as follows. Section 2 gives some related work. Section 3 introduces the basic idea of *k*-clique community tree and closed *l*-*s*-clique community. Section 4 describes our algorithms KCliqueTree and LSBridge. Section 5 conducts case study on DBLP dataset. Section 6 concludes the paper.

2. Related Work

Community detection has gained great focus since the last decade. And there are extended work in this field [5-9]. Several milestone definitions are given to describe the concept of community. And Clique community [25] is one of the earliest definitions, which is widely used in social network analysis. Clique community is directly from the concept of clique in discrete mathematics. Recently, Palla [26] gave the definition of *k*-clique community based on the definition of *k*-clique. *K*-clique community is a definition focusing on the overlapping regions between communities. There is extended work on the detection of *k*-clique community [12-17]. However, the research on the hierarchical structure of *k*-clique community and its bridging member is rare.

A well known community detection method called hierarchical clustering [18] is commonly used to explore the hierarchical structure of graph. A definition on the distance between vertices is given first. Then the distance between all vertice pairs is computed. At last, a hierarchical tree is generated according to the distance with a agglomerative algorithm or divisive algorithm. The hierarchical tree displays several levels of grouping of the vertices with small clusters included within large clusters [27]. Note that in this paper, we explore the hierarchical structure of *k*-clique community and cluster indicates *k*-clique community in our work**.** There is no manually defined similarity (distance) either.

3. *K*-clique Community Tree and Closed *l*-*s*-clique Community

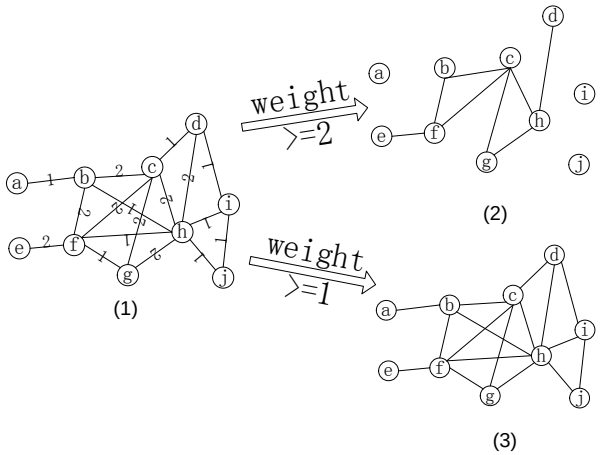


Figure 2. Graph Generated by Constraint with Constraint Dimension

Before further discussion, let us look at example 2. Note that a specified constraint (The papers should be published in 2003) is used during the abstraction in example 1. In this paper, a constraint is a set of constraint dimensions. A constraint dimension is an atomic rule for generating the graph. The constraint in example 1 is composed only one constraint dimension: 'The papers should be published in year *Y*'. To make the specified constraint 'The papers should be published in 2003' tighter, constraint dimension 'the authors co-published at least *N* times' could be appended. Then if *N* is set to be 2, they together compose a new specified constraint 'The papers should be published in 2003, the authors co-published at least 2 times'.
Example 2: Given the constraint 'The papers should be published in 2004', a tiny subset of DBLP is abstracted as graph (1) in left Figure 2. Here the number on an edge indicates the times for the coauthorship of the vertices for the edge. Then constraint dimension 'the authors

co-published at least *N* times' is appended to construct a new constraint. If *N* is set to be larger than or equal to 2, the subset of DBLP is abstracted as graph (2) in right upper Figure 2; if *N* is set to be larger than or equal to 1, the subset of DBLP is abstracted as graph (3) in right

bottom Figure 2. In graph (2), *b, c* and *f* are in a 3-clique community; *c, g* and *h* are in a 3-clique community. In graph (3), *b, c, d, f, g, h, i, j* are in a 3-clique community. The 3-clique community in graph (3) could be viewed as the first level community in describing the strength of the vertices connections; the two 3-clique communities in graph (2) could be viewed as the second level communities in describing the strength of the vertices connections. Then *b, f, g, h* could be viewed as playing the role of bridging different communities.

According to example 1 and 2, *k*-clique community could be viewed as the basis for analyzing the hierarchical structure and bridging member of *k*-clique community: (1) the value of *k* reflects the strength of the vertices connections inside a *k*-clique community: the larger *k* is, the higher the strength is; (2) the value of *k* indicates a subset relationship, *i.e.*, a larger *k*-clique community is inside a smaller *k*-clique community; (3) Studying the *k*-clique communities helps to find the vertices playing the bridging role.

3.1. Terminologies

Let $p_i(1 \square i \square n)$ be a constraint dimension where *n* is a positive integer. The domain for the parameter θ of p_i is a finite set $\{\theta_1, \theta_2, \theta_3, \ldots, \theta_i, \ldots, \theta_j, \ldots \theta_n\}$, where $\theta_i < \theta_j$ if $i < j$. Then a specified constraint could be denoted as $SP=\{sp_i | sp_i = (\theta_{low}, \theta_{high})\}$ $(1 \square i \square n)$. *SP* is a finite nonempty set. Here sp_i is a specified constraint on constraint dimension p_i. With a specified constraint *SP*, an unweighted undirected graph abstracted from real data is denoted as $G = (V, E)$, where *V* and *E* are both finite nonempty set. The subgraph induced by vertice set *Vs* is denoted as $G|_{Vs}$. Whether $G|_{Vc}$ is a *k*-clique community is judged by boolean function k-Clique(*Vs, G, k*). If $G|_{Vs}$ is a *k*-clique community, k-Clique(*Vs, G, k*)=1; else k-Clique(*Vs, G, k*)=0.

Now, let us revisit example 2. For graph (2) in example 2, $SP=\{sp_1, sp_2\}$, sp_1 = 'The papers should be published in 2004', sp_2= 'the authors co-published at least 2 times', p1='The papers should be published in year Y', p2= 'the authors co-published at least *N* times'. We call graph (2) is generated with sp_2 from graph (1) or graph (2) is generated with constraint SP from original data.

3.2. *K*-Clique Community Tree and Closed *l-s*-clique Community

Definition 1: A *k-clique community tree* is a tree structure generated with specified constraint *SP*, whose vertices are *k*-clique communities. It could be constructed by the following steps:

1) Generate a node for each *k*-clique community C_k detected with constraint *SP*;
2) For *k*-clique community C_k and *k*+1-clique community C_{k+1}, if C_{k+1} is a subset of C_k, the node for C_{k+1} is a child of the node for C_k;
3) The root is the *k*-clique community with the smallest *k*. It limits the scope of the tree.

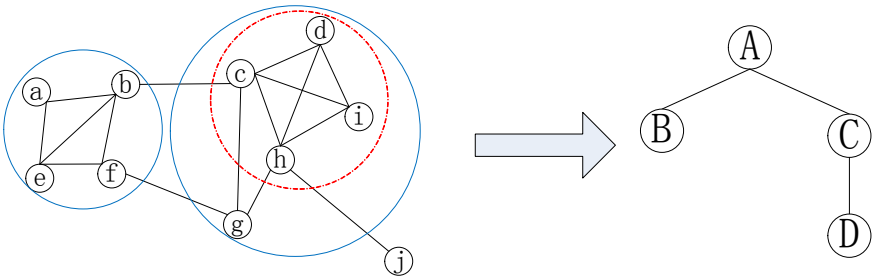


Figure 3. An Example for *k*-clique Community Tree

Example 3: The *k*-clique community tree in the right part of Fig. 3 is for the graph in the left part of Fig.3.

Node *A* is the 2-clique community composed of {*a*, *b*, *c*, *d*, *e*, *f*, *g*, *h*, *i*, *j*}. Node *B* is the 3-clique community composed of {*a*, *b*, *e*, *f*}. Node *C* is the 3-clique community composed of {*c*, *d*, *g*, *h*, *i*, *j*}. Node *D* is the 4-clique community composed of {*c*, *d*, *h*, *i*}. *B* and *D* are both leaves of the tree: the strength vertices connect to each other inside them is relatively strong. Vertices *b* and *c* are members of two children of the same parent node: they play the bridging role.

According to example 3, *k*-clique community tree could be used as an index for exploring *k*-clique community: if users are interested in the communities inside which the strength of vertices connections are strong, they could check the leaves of the tree; if users are interested in the bridging members, they could check the nodes with many children.

To describe the bridging members of *k*-clique community, the concept of *l*-*s*-clique community pivot is introduced.

Definition 2: Let $l\text{Clique}_1(Vl_1, El_1)$ and $l\text{Clique}_2(Vl_2, El_2)$ be two *k*-clique communities ($k=l$), vertice *v* is a *l*-*s*-*clique community pivot* of $l\text{Clique}_1$ and $l\text{Clique}_2$ IFF there exists at least one *k*-clique community ($k=s$ and $l>s$) $s\text{Clique}_1(Vs, Es)$ which has a connected subgraph *linkG*(*linkV*, *linkE*) satisfying the following conditions:

(1) $Vl_1 \subset Vs$, $Vl_2 \subset Vs$;

(2) $v \in linkV \cap (Vl_1 \cup Vl_2)$;

(3) $linkV \cap Vl_1 \neq \varnothing$ and $linkV \cap Vl_2 \neq \varnothing$;

(4) For each edge *uw* in *linkE*, neither $\{u, w\} \subset Vl_1$ nor $\{u, w\} \subset Vl_2$ holds.

Whether a vertice *v* in *G* is a *l*-*s*-clique community pivot is judged by boolean function l-s-bridge(*v*, *G*, *l*, *s*). If *v* is an *l*-*s*-clique community pivot, l-s-bridge (*v*, *G*, *l*, *s*) = 1; else l-s-bridge (*v*, *G*, *l*, *s*) = 0.

According to the definition of *l*-*s*-clique community pivot, for *k*-clique communities ($k = l$) $l\text{Clique}_1(Vl_1, El_1)$ and $l\text{Clique}_2(Vl_2, El_2)$, the vertices in $Vl_1 \cup Vl_2$ are all *l*-*s*-clique community pivots. Let us look at an example.

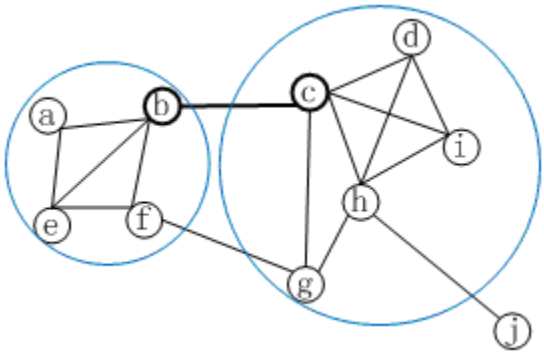


Figure 4. An Example for *k*-clique Community Pivot

Example 4: In Figure 4, *a*, *b*, *c*, *d*, *e*, *f*, *g*, *h*, *i* and *j* compose a 2-clique community; *a*, *b*, *e* and *f* compose a 3-clique community; *c*, *d*, *g*, *h*, *i* and *j* compose a 3-clique community. The vertice *b* could be viewed as a *l*-*s*-clique community pivot (*l*=3, *s*=2) between the 3-clique community composed by {*a*, *b*, *e*, *f*} and the 3-clique community composed by {*c*, *d*, *g*, *h*, *i*, *j*}: there exists a 2-clique community composed by {*a*, *b*, *c*, *d*, *e*, *f*, *g*, *h*, *i*, *j*} which has a connected subgraph {{*b*, *c*}, {*bc*}} satisfying the 4 conditions above: 1) {*a*, *b*, *e*, *f*}⊂{*a*, *b*, *c*, *d*, *e*, *f*, *g*, *h*, *i*, *j*}, {*c*, *d*, *g*, *h*, *i*, *j*}⊂{*a*, *b*, *c*, *d*, *e*, *f*, *g*, *h*, *i*, *j*}; 2) $b \in \{b, c\} \cap (\{a, b, e, f\} \cup \{c, d, g, h, i, j\})$; 3) $\{b, c\} \cap \{a, b, e, f\} \neq \varnothing$ and $\{b, c\} \cap \{c, d, g, h, i, j\} \neq \varnothing$; 4) for the edge *bc* in {*bc*}, {*b*, *c*}⊂{*a*, *b*, *e*, *f*} does not hold, {*b*, *c*}⊂{*c*, *d*, *g*, *h*, *i*, *j*} does not hold.

According to example 3 and example 4, on a *k*-clique community tree, *l*-*s*-clique community pivot describes how children nodes connect to each other inside their parent node. And according to example 3, the larger the *k*-clique community is, the higher level it has on the *k*-clique community tree. To simplify the *k*-clique community tree while avoiding the loss of information for *k*-clique community, *l*-*s*-clique community is introduced.

Definition 3: Given a graph *G* and a vertice set *Vs*, $G|_{Vs}$ is an *l*-*s*-clique community if it satisfies the following conditions:

(1) There exists $s \geq 2$ satisfying k-clique(*Vs*, *G*, *s*)=1;

(2) There exists $Vl \subset Vs$ ($l \geq s$) satisfying k-clique(*Vl*, *G*, *l*)=1;

(3) $|Vl|/|Vs| > \alpha$ ($1 \geq \alpha > 0$, α is specified by the user manually).

Whether $G|_{Vs}$ is a *l*-*s*-clique community is judged by boolean function l-s-clique(*Vs*, *G*, *l*, *s*, α). If $G|_{Vs}$ is an *l*-*s*-clique community, l-s-clique(*Vs*, *G*, *l*, *s*, α) = 1; else l-s-clique (*Vs*, *G*, *l*, *s*, α) = 0.

Three conclusions could be drawn according to definition 3:

(1) An *l*-*s*-clique community is also a *k*-clique community and *l*-*s*-clique community could be represented by a node of *k*-clique tree;

(2) When *l* is equal to *s*, a *k*-clique community is also an *l*-*s*-clique community;

(3) When *l* is larger than *s*, a *k*-clique community may be an *l*-*s*-clique community: the larger α is, with the smaller possibility a *k*-clique community becomes an *l*-*s*-clique community. An extreme case is when α is equal to 1, only clique community is *l*-*s*-clique community.

According to the conclusions above, the definition for *l*-*s*-clique community is more relaxed than clique community while tighter than *k*-clique community. Therefore, given a graph, the count for *l*-*s*-clique community may be large. And for some *l*-*s*-clique communities, their sub-communities are also *l*-*s*-clique communities. To lay emphasis on the extraordinary nature of the leaves for a *k*-clique community tree, closed *l*-*s*-clique community is introduced.

Definition 4: Given a graph G and a vertice set Vs, $G|_{Vs}$ is a closed l-s-clique community if it satisfying the following conditions:
(1) $l>s\geq 2$ and l-s-clique(Vs, G, l, s, α)=1;
(2) If $l_i>l$, then l-s-clique(Vs, G, l_i, s, α)=0;
(3) If $s_i<s$, then l-s-clique(Vs, G, l, s_i, α)=0;
(4) There does not exist a vertice set Vc satisfying $Vs\subset Vc$ and l-s-clique(Vc, G, l, s, α)=1.

Note that definition 4 indicates only a leaf of k-clique community tree could be a closed l-s-clique community.

Example 5: After applying the concept of l-s-clique community, the k-clique community tree on Fig. 3 is transformed into the one on right Figure 5. Node A is the 2-clique community composed of {a, b, c, d, e, f, g, h, i, j}. Node B is the 3-clique community composed of {a, b, e, f}. Node C is the l-s-clique (l=4, s=3, α<0.8) community composed of {c, d, g, h, i, j}. B and C are both leaves of the tree: the strength vertices connect to each other inside them is relatively strong. Vertices b and c are still members of two children of the same parent node: they play the bridging role.

According to example 5, after introducing l-s-clique community, when l is bigger than s, α is equal or close to 1, l-s-clique community is highly similar to the k-clique community (l□k□s) it contains. This keeps the index function of k-clique tree. Besides simplifying the k-clique tree, l-s-clique community helps to avoid regenerating highly similar k-clique communities, which reduces the time cost.

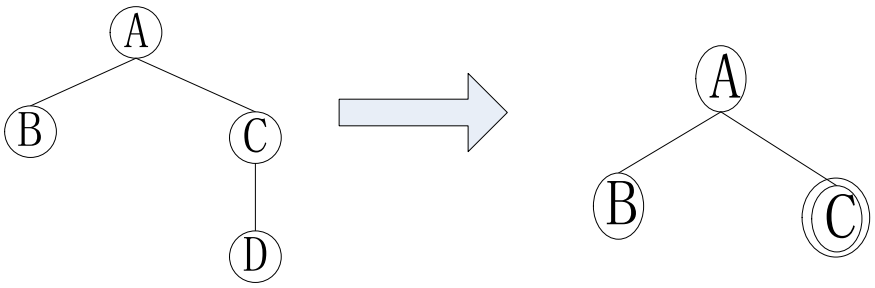


Figure 5. An Example for *k*-clique Community Tree after Introducing *l*-*s*-Community

3.3. The Property of *l*-*s*-community

Properties similar to the famous Apriori property [31] exist for l-s-community. They are expressed as the following two theorems.

Theorem 1: Given l_i, l_j and s ($l_i>l_j>s$) and given vertice set Vs (l-s-clique(Vs, G, l_i, s, α)=1), there must exists a vertice set Vc satisfying $Vc\subset Vs$ and l-s-clique(Vc, G, l_j, s, α)=1.
Proof: According to the definition of l-s-community, there musts exist a vertice set Vs_k satisfying $Vs_k\subset Vs$, $|Vs_k|/|Vs|>\alpha$ and k-clique(Vs_k, G, l_i)=1. For vertice set Vc ($Vc\subset Vs$), there exist two cases to be discussed:
a) if k-clique(Vc, G, l_j)=0, then l-s-clique(Vc, G, l_j, s, α)=0;
b) if k-clique(Vc, G, l_j)=1, since $l_i>l_j$, according to the definition of k-clique community, we can get $|Vc|>|Vs_k|$ and $|Vc|/|Vs|>\alpha$, thus l-s-clique(Vc, G, l_j, s, α)=1.
The second case must exist. And $Vs_k=Vc$ is an example. Thus Theorem 1 holds.

Theorem 2: Given k, s_i, s_j ($l>s_i>s_j$) and given vertice set Vs (l-s-clique(Vs, G, k, s_j, α)=1), there must exists a vertice set Vc satisfying $Vc \subset Vs$ and l-s-clique(Vc, G, k, s_i, α)=1.
Proof: According to the definition of *l-s*-community, there musts exist a vertice set Vs_k satisfying $Vs_k \subset Vs$, $|Vs_k|/|Vs|>\alpha$ and k-clique(Vs_k, G, s_i)=1. For vertice set Vc ($Vc \subset Vs$), there exist two cases to be discussed:
a) if k-clique(Vc, G, s_j)=0, then l-s-clique(Vc, G, l, s_i, α)=0;
b) if k-clique(Vc, G, s_j)=1, since $s_j > s_j$, according to the definition of k-clique community, we can get $|Vs|>|Vc|$ and $|Vs_k|/|Vc|>\alpha$, thus l-s-clique(Vc, G, l, s_j, α)=1.
The second case must exist. And $Vs_k=Vc$ is an example. Thus Theorem 2 holds.
Theorem 1 and theorem 2 together provide a path to detect *l-s*-community. According to theorem 1, given the s for k-clique community, the closed *l-s*-clique community could be discovered by checking the k-clique community with specified l. The space is searched by adjusting k (from bigger to smaller) and the search stops once discovering *l-s*-community or l□s; according to theorem 2, given the l for k-clique community, the closed *l-s*-clique community could be discovered by checking the k-clique community with specified s. The space is searched by adjusting s (from smaller to bigger) and the search stops once discovering *l-s*-community or l□s.

3.4. *K*-clique Community Dimension Tree

To describe the hierarchical structure of k-clique community on constraint dimension p_i, we introduce the concept of k-clique community dimension tree. The discussion is based on the assumption that with the decreasing of θ from θ_n to θ_1, the number of edges of G also decreases.

Definition 5: A *k-clique community dimension tree* is a tree structure generated with specified constraint SP and constraint dimension p_i, whose vertices are k-clique communities. Here SP does not contain sp_i. It could be constructed by the following steps:
(1) Generate a node for each k-clique community C_k detected with constraint $SP \cup \{sp_i = (\theta_i, \theta_i)\}$;
(2) For k-clique community C_i^k with constraint $SP \cup \{sp_i = (\theta_i, \theta_i)\}$ and k-clique community C_{i+1}^1 with constraint $SP \cup \{sp_i = (\theta_i, \theta_i)\}$, if C_{i+1}^1 is a subset of C_i^k, the node for C_{i+1}^1 is a child of the node for C_i^k;
(3) The root is the k-clique community generated with the smallest θ (θ_1). It limits the scope of the tree.

According to definition 5, the expression of k-clique community dimension tree is similar to k-clique community tree. However, the semantic is different: the k value for k-clique community dimension tree is permanent, the hierarchical structure is generated by the different strength of the constraint; the hierarchical structure of k-clique community tree is generated by different k

The algorithm to generate k-clique community dimension tree is the same as that to generate k-clique community tree. Therefore, in the next section, we will only focus on the algorithm for k-clique community tree.

4. The *k*-clique Community Tree Construction Algorithm and the *k*-clique Community Pivot Detection Algorithm

4.1. Sketch of KCliqueTree

According to the analysis above, KCliqueTree is proposed to construct *k*-clique community tree, which is shown in Algorithm 1. Here both *k*-clique community and closed *l*-*s*-clique community are considered excluding non-closed *l*-*s*-clique community. KCliqueTree is roughly divided into two steps:

1) The first step is shown by line 1 to line 6. Through searching the maximum clique community for *kclique*, it gives the upper limit for the search in KCliqueSubTree and LSCliqueSubTree. If *kclique* is a clique community, the algorithm stops.

2) The second step is shown by line 7 to line 19. If the leaf does not indicate a closed *l*-*s*-community, KCliqueSubTree is called; if the leaf indicates a closed *l*-*s*-community, both KCliqueSubTree and LSCliqueSubTree are called; the nodes generated by KCliqueSubTree indicate small *k* (here *k* is set to be smaller than 5) while the nodes generated by LSCliqueSubTree indicate large *k*.

Algorithm 1 KCliqueTree

A CSV plot [17] based algorithm to construct *k*-clique community tree

Input: *G*, *csvPlotK(*CSV plot for *G)*, *kClique*(a *k*-clique community), *k*(the value for *k* of *kClique*), *lsCal*(the flag for the existence of *l*-*s*-community), α
Output: *tree* (a *k*-clique community tree with *kClique* as the root)

```
1.  maxClique = MaxCliqueCSV(G, csvPlotK, k);
2.  l_max = |maxClique|;
3.  if l_max/|kClique |>α
4.     treeRoot = {k, l_max, kClique}; return;
5.  endif
6.  store {k, l_max, {maxClique}, kClique} in a temporary buffer;
7.   treeRoot = kClique;
8.   if lsCal = false
9.      tree = KCliqueSubTree(treeRoot, kClique, G, csvPlotK, k, l_max);
10.  else
11.     if k < 5
12.        headTree = KCliqueSubTree(treeRoot, kClique, G, csvPlotK, k, 5);
13.     else
14.        headTree = treeRoot;
15.        for each leafNode in headTree
16.           LSCliqueSubTree(leafNode, the k-clqiue communities inside leafNode,
      G, csvPlotK, k, l_max, α, True);
17.        end
18.     endif
19.  endif
```

Here KCliqueSubTree is simple. It is divided into two steps:

1) The first step is to call KCliqueCSV [17], perform a transverse and check out all the *k*-clique communities with different *k*.

2) The second step is to construct a hierarchical tree according to subset relationship between *k*-clique communities

We will lay emphasis on LSCliqueSubTree, which is shown in Algorithm 2.

LSCliqueSubTree is a recursive algorithm based on theorem 1 and theorem 2. It uses the buffer to store the intermediate results of computation, reduces the computation by calling subprocedure calOrLoad, saveRelation and reduces the search scope of the next step by calling subprocedure subG, subCSVPlot.

Similarity computation is an essential step for memory based method. The most commonly used similarity computation formulas are Pearson coefficient, cosine similarity and adjust cosine similarity. Here Pearson coefficient is adopted for improvement. Our method also works for other similarly computation formulas. It is roughly divided into two steps:

1) The first step is shown by line 1 to line 6. It judges whether the recursive call should be stopped.
2) The second step is shown by line 7 to line 27, which could be divided into three sub-steps:

Algorithm 2 LSCliqueSubTree

A CSV plot [17] based algorithm to construct *k*-clique community tree whose leaves are closed *l*-*s*-clique community

Input: *treeRoot*, *G*, *sClique*(a *k*-clique community with *s*), *csvPlotS(*CSV plot for *sClique)*, *s*, *l*, *α* for *l*-*s*-clique community, *maxCal*(flag)

Output: *tree* (a *k*-clique community tree with *sClique* as the root and closed *l*-*s*-clique communities as its leaves)

```
1.  if s>l
2.     return;
3.  endif
4.  if (treeRoot is a l-s-clique community) or (treeRoot has children)
5.     return;
6.  endif
7.   if s==l
8.     treeRoot = {s, l, wclique}; return;
9.  endif
10. if maxCal==true
11.    sG = subG(G, sClique);
12.    sCSVPlotS = subCSVPlot(csvPlotS, sClique);
13.    maxClique = MaxCliqueCSV(sG, sCSVPlotS, s);
14.    if |maxClique| < l
15.       l = |maxClique|;
16.    endif
17. endif
18. lKCliques = calOrLoad(sClique, l);
19. calContinue = true;
20. for each lClique in lKCliques
21.     if |lClique|/|sClique| >α
22.       make {s, l, sClique} as a child of treeRoot;
23.       calContinue = false;
```



```
24.      endif
25.  endfor
26.  if calContinue == false
27.     return;
28.  endif
29.  lKCliques = calOrLoad(sClique, s+1);
30.  for each sPlusClique in sPlusKCliques
31.     saveRelation(sPlusClique, lKCliques, s+1, l);
32.     make sPlusClique a child of treeRoot and denote it as subRoot;
33.     LSCliqueSubTree(subRoot, sPlusClique, sG, sCSVPlotS, s+1, l,
        α, true);
34.  endfor
35.  nextL = findNextMax(csvPlotS, l_max);
36.  LSCliqueSubTree(treeRoot, sClique, sG, sCSVPlotS, s, nextL, α,
        false);
```

① The first step is shown by line 10 to 17. If *maxCal* is true, the search the maximum clique community inside *sClique* and update the upper limit for the searching.

② The second step is shown by line 18 to 28. It first judges whether there exists a *l*-*s*-clique community with *s* and *l* inside *sClique*. If it is true, this community is a closed *l*-*s*-clique community according to theorem 2; the results are stored and the recursive call is stopped.

③ The third step is shown by line 29 to 36. It updates *sClique* as a *k*-clique community (*k*=*s+1*), *s* as *s+1* for the *l*-*s*-clique community and performs recursive call; or updates *l* as *nextL* for the *l*-*s*-clique community and performs recursive call.

The subprocedure calOrLoad appears twice on line 11 and line 19 respectively. On line 11, it checks whether there exists the information for *k*-clique community with *l* inside *sClique* in the buffer, if the result is true, it gets the information from the buffer; or call KCliqueCSV [32] and store the results into *lKCliques* and store {*sClique*, *lKCliques*, *s*, *l*} into the buffer.

The subprocedure saveRelation appears on line 21. It first searches the parent-child relationship *k*-clique communities with *s+1* and *l* inside *sClique*, then store them into the buffer.

Algorithm 3 LSBridge

A CSV plot [17] based algorithm to detect *l*-*s*-clique community pivot

Input: *G*, *csvPlotK(*CSV plot for *G)*, *kClique*(a *k*-clique community), *k*(the value for *k* of *kClique*), *lsCal*(the flag for the existence of *l*-*s*-community), *α*

Output: *tree* (a *k*-clique community tree with *kClique* as the root)

```
1.  lsBridge = ∅;
2.  if l>s
3.     lCliqueG1=subG(G, lClique1);
4.     lCliqueG2=subG(G, lClique2);
5.     sCliqueG=subG(G, sClique) ;
6.     LSBridge=LSBridge∪(lClique1∩ lClique2);
7.     deleteSubG(sCliqueG, lCliqueG1∪lCliqueG2 ) ;
8.     linkedGraphs=subLinkedG(sCliqueG);
```

```
9.      for each linkedG in linkedGraphs
10.         if (linkedG∩ lCliqueG1≠∅)&&(linkedG∩ lCliqueG2≠∅)
11.       LSBridge=LSBridge ∪ Vertice(linkedG) ∩ (Vertice(lClique1)
      ∪Vertice(lClique2));
12.         endif
13.     endfor
14.  endif
```

4.2. Sketch of LSBridge

Given *lClique1* and *lClique2* (*k*-clique communities with *l*), their *l*-*s*-clique community pivot could be searched by method of exclusion. The strategies for exclusion are as follows:

1) If there exists a *l*-*s*-clique community pivot, *lClique1* and *lClique2* exist in at least one *k*-clique community with *s* denoted as *sClique1*;
2) If the connected graph *linkG*, which helps to find the *l*-*s*-clique community pivot, does not share vertices with *lClique1* and *lClique2*, *linkG* is neither *lClique1* nor *lClique2*;
3) A shared vertice of *lClique1* and *lClique2* is an *l*-*s*-clique community pivot.

According to the definition of *l*-*s*-clique community pivot and the strategies for exclusion, LSBridge is proposed to search the *l*-*s*-clique community pivot of two *k*-clique communities with two *l*. This algorithm reduces the time cost by filtering with CSV plot. As shown in algorithm 3, LSBridge is roughly divided into two steps:

1) The first step is shown by line 3 to 5, it generates the subgraphs *lClique1*, *lClique2* and *sClique* are in respectively.
2) The second step is shown by line 6 to 14. Line 6 finds the shared vertices of *lClique1* and *lClique2*. Line 7 to line 11 checks whether the non shared vertices of *lClique1* and *lClique2* are *l*-*s*-clique community pivots based on exclusion strategy 2.

4.3. Time complexity

For KCliqueTree, the time complexity highly depends on SCP and LargeKCliqueCSV: if the leaves are not *l*-*s*-clique communities, the time cost is the cost for traversing all the *k*-clique communities with different *k*; if the leaves are *l*-*s*-clique communities, the smaller αis, the number of *k*-clique communities to be traversed is smaller and the time cost is less. In [17], Xu has demonstrated the high efficiency of LargeKCliqueCSV to detect large *k*-clique communities. Thus KCliqueTree is also efficient when *k* is large.

The scale of the input *k*-communities for LSBridge is very small. The worst time complexity of subG, subLinkedG and deleteSubG is O(|E|). Thus LSBridge is linear.

Since the time complexity for both KCliqueTree and LSBridge is not a problem, the focus will be on the results generated by the two algorithms.

5. Case study

All experiments are conducted on an INTEL core 2DuoProcessorE2160 with 2G memory, running Windows XP. All algorithms are implemented with C++ and STL.

5.1. The Dataset

Since the loss of information is significant for SMD [33] dataset. Only DBLP dataset is adopted for case study. DBLP is an integrated database system composed of the

bibliography information in the field of computer science. And it is stored as XML file. In the graph transformed from DBLP data, vertices represent authors while edges represent the co-authorship. The existence of an edge between two authors indicates that in a given time period, the times for their co-authorship is not smaller than a threshold. When the threshold is set to be 2, the statistical information for DBLP dataset is shown in Figure 6 and Figure 7. Figure 6 and Figure 7 show that the bigger the year is, the larger $|V|$ and $|E|$ are. But the average $|E|/|V|$ is always small.

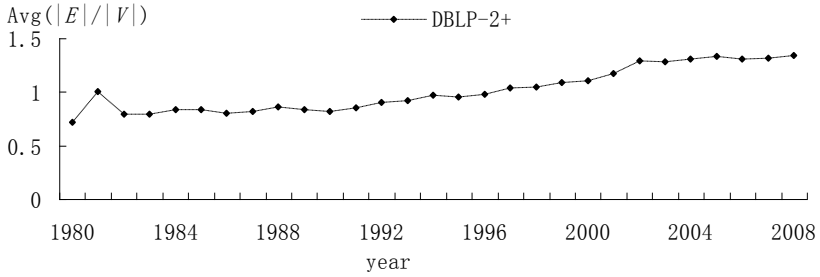


Fig. 6. The statistical information on average |E|/|V| for DBLP dataset

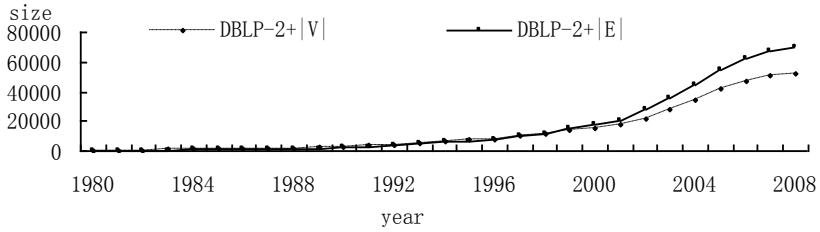


Figure 7. The Statistical Information on Vertices and Edges for DBLP Dataset

5.2. Evaluation and Results

The constraint for the graph abstraction is set to be 'the papers are published in year *Y*, the authors co-published at least twice'. The *k*-clique community tree generated from the abstracted graph will be studied. According to the structure of *k*-clique community trees, they are classified into two categories: balanced tree and non-balanced tree. Our case study will focus on the trees whose roots are 3-clique communities.

5.2.1. Case 1

When *Y* is set to be 2006, a *k*-clique community composed of 22 vertices is discovered. The time cost for searching the *l*-*s*-clique community pivot is 0.11 seconds. When α is set to be 0.8, the time cost for constructing the *k*-clique community tree is 0.125 seconds. As shown in Figure 6, the tree is balanced.

A is a 3-clique community composed of 22 vertices. *F* is a 7-4-clique community composed of 7 vertices. And the other nodes are 4-clique community. When *l* is set to be 4 and *s* to be 3, there are 8 *l*-*s*-clique community pivots inside *A*. The detail is shown in Table 1. Table 1 shows that 1) *B*, *C*, and *E* share relatively more *l*-*s*-clique community pivots; 2) *D* and *F* share relatively more *l*-*s*-clique community pivots.

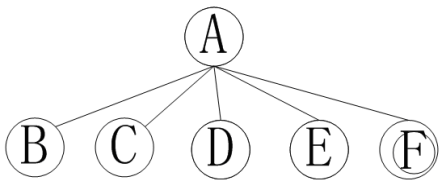


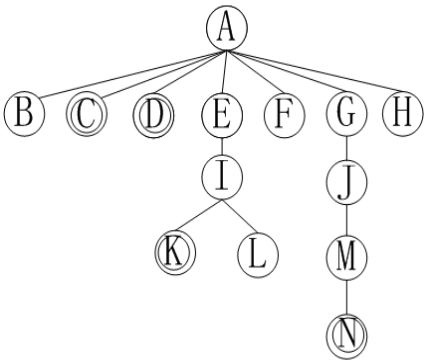
Figure 6. An Example for Balanced *k*-clique Community Tree

Table 1. The *l*-*s*-clique Community Pivots for the Tree in Figure 6

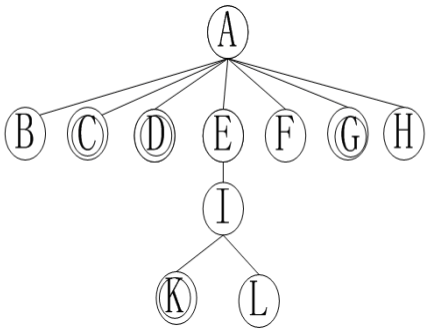
Node	*l*-*s*-clique community pivots	The communities bridged
B	Jiawei Han, Philip S. Yu, Jian Pei	*C, D, E, F*
C	Jiawei Han, Philip S. Yu, Xifeng Yan	*B, D, E, F*
D	Bing Liu, Wei Wang, Baile Shi	*B, C, E, F*
E	Philip S. Yu, Jian Pei	*B, C, D, F*
F	Jian Pei, Wei Wang, Baile Shi, Ada Wai-Chee Fu	*B, C, D, E*

The analysis above shows that: 1) As the *l*-*s*-clique community pivots, Jiawei Han, Jian Pei, Philip S. Yu, Bing Liu, Wei Wang are all famous scholars in the field of Data Mining: 2) In general, the authors represented by vertices which are not *l*-*s*-clique community pivots are unnamable scholars.

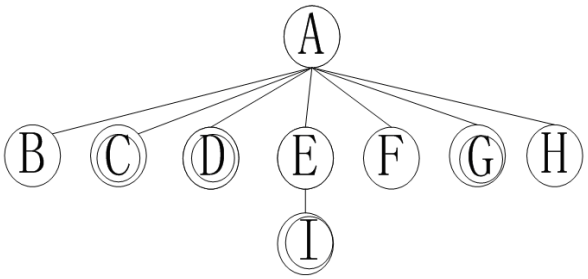
5.2.1. Case 2



(a) *l*-*s*-clique communities(*α*=0.8)



(b) *l*-*s*-clique communities(*α*=0.7)



(c) *l*-*s*-clique communities (*α*=0.6)

Figure 7. An Example for Non-Balanced *k*-clique Community Trees

When *Y* is set to be 2008, a *k*-clique community composed of 49 vertices is discovered. The time cost for searching the *l*-*s*-clique community pivot is 0.266 seconds. When α is set to be 0.8, the time cost for constructing the *k*-clique community tree is 0.281 seconds and the tree is non-balanced; When α is set to be 0.7, the time cost for constructing the *k*-clique community tree is 0.235 seconds and the tree is non-balanced; When α is set to be 0.6, the time cost for constructing the *k*-clique community tree is 0.219 seconds and the tree could be regarded as balanced. The results are shown in Figure 7.

When α is reset from 0.8 to 0.7, the subtree whose root is *G* changes most. The reason is that *G* is composed of 10 vertices and 8 of them compose a clique community. When α is reset from 0.7 to 0.6, the subtree whose root is *I* changes most. The reason is that *G* is composed of 14 vertices: 8 of them compose *L* (a 6-clique community) while 9 of them compose *K* (a clique community).

Table 2. The *l*-*s*-clique Community Pivots for the Trees in Figure 7

Node	*l*-*s*-clique community pivots	The communities bridged
B	Thomas Sikora, Lutz Goldmann	*C, D, E, F, G, H*
C	Vasileios Mezaris, Ioannis Kompatsiaris	*B, D, E, F, G, H*
D	Thomas Sikora, Noel E. O'Connor, Lutz Goldmann, Tomasz Adamek,Yannis S. Avrithis	*B, C, E, F, G, H*
E	Thomas Sikora, Noel E. O'Connor,	*B, C, D, F, G, H*
F	Vasileios Mezaris, Ioannis KompatsiarisAlan	*B, C, D, E, G, H*
G	Tomasz Adamek, Yannis S. Avrithis, Evaggelos Spyrou	*B, D, E, H*
G	Vasileios Mezaris, Ioannis Kompatsiaris	*C, F*
H	Noel E. O'Connor, Yannis S. Avrithis	*B, C, D, E, F, G*

When *l* is set to be 4 and *s* to be 3, there are 10 *l*-*s*-clique community pivots inside *A*. The detail is shown in table 2. Table 2 shows that 1) *C* and *F* share *l*-*s*-clique community pivots; 2) *B, D, E* and *H* share relatively more *l*-*s*-clique community pivots; 3) The vertices inside *G* could be classified into two types of *l*-*s*-clique community pivots: bridging *C&F* and bridging *B, D, E&H*.

The analysis above shows that: 1) The *k*-clique community detected here is for the field of multimedia: 2) In general, the authors represented by vertices which are not *l*-*s*-clique community pivots published at a relatively later date.

The analysis on case 1and case 2 shows that: 1) Searching *l*-*s*-clique community pivots helps to discover interesting knowledge; 2) *k*-clique community tree helps to understand how the vertices are bridged by key vertices and then construct a community.

5.2.3. Case 3

Two constraint dimensions are chosen: 'The papers should be published in year *Y*' and 'the authors co-published at least *N* times'. After specifying *Y*, we will analyze the *k*-clique community dimension tree on dimension 'the authors co-published at least *N* times'. Here *N* is set to be 2 and 3 respectively. Then there are at most two levels for the *k*-clique community dimension tree. Since it is too simple, no visualization is used here.

Considering the root node *A* in Figure 6, if *N* is set to be 3, it has 4 children. Like *l*-*s*-clique community pivots, the authors bridging the four children are Jiawei Han, Jian Pei and Philip S. Yu.

Considering the root node *A* in Figure 7, if *N* is set to be 3, it has 5 children. Like *l*-*s*-clique community pivots, the authors bridging the 5 children are Ioannis Kompatsiaris, Noel E. O'Connor and Yannis S. Avrithis.

According to the analysis above, although *k*-clique community dimension tree and *k*-clique community tree are different from each other on the contents, they are similar to each other in describing bridging members.

6. Conclusions

In this paper, we define *k*-clique community tree and closed *l*-*s*-clique community to be the index for analyzing *k*-clique communities in a graph abstracted from real data. *K*-clique community dimension tree is also discussed here. We also define *l*-*s*-clique community pivot to describe the bridging members of *k*-clique community. KCliqueTree is proposed to construct the *k*-clique community tree and LSBridge is proposed to search the *l*-*s*-clique community pivots. Since our algorithms take advantage of LargeKCliqueCSV, their time costs are very low. The case study on DBLP verifies the availability of our definitions and algorithms and their low time costs. Obvious our definitions and algorithms can be applied in analyzing data in various fields such as email contact, mobile contact and internet browsing.

References

[1] L. C. Freeman, "The sociological concept of 'group': An empirical test of two models", American Journal of Sociology, vol. 98, no. 1, **(1992)**.
[2] http://www.informatik.uni-trier.de/~ley/db/.
[3] M. E. J. Newman, "The structure and function of complex networks", SIAM Review, vol. 45, **(2003)**.
[4] L. F. Costa, F. A. Rodrigues, G. Travieso and P. R. Villas Boas, "Characterization of complex networks: A survey of measurements", Advances In Physics, vol. 56, **(2007)**.
[5] E. Ravasz, A. L. Somera, D. A. Mongru, Z. Oltvai and A. L. Barab´asi, "Hierarchical organization of modularity in metabolic networks, Science, vol. 297, **(2002)**.
[6] A. Arenas, A. Fernández, S. Fortunato and S. Gómez, "Motif-based communities in complex networks", Journal of Physics A: Mathematical and Theoretical, vol. 41, 224001, **(2008)**.
[7] S. Fortunato, "Community detection in graphs", Physics Reports, vol. 486, **(2010)**, pp. 3–5.
[8] K. K. Xu, C. J. Tang, C. Li, Y. X. Jiang and R. Tang, "An MDL Approach to Efficiently Discover Communities in Bipartite Network", DASFAA, Japan, vol. 1, **(2010)**.
[9] C. Aggarwal, Y. Xie and P. Yu, "Towards community detection in locally heterogeneous networks", SDM, USA, **(2011)**.
[10] M. Girvan and M. E. J. Newman, "Community structure in social and biological networks", Proc. Natl. Acad. Sci. USA, vol. 99, **(2002)**.
[11] R. Guimerà, M. Sales-Pardo and L. A. N. Amaral, "Module identification in bipartite and directed networks", Phys Rev E, vol. 76, **(2007)**.
[12] I. Derényi, G. Palla and T. Vicsek, "Clique percolation in random networks", Phys. Rev. Lett, vol. 94, 160202, **(2005)**.
[13] P. Hui, E. Yoneki and S. Y. Chan, "Distributed Community Detection in Delay Tolerant Networks", Proceedings of 2nd ACM/IEEE international workshop on Mobility in the evolving internet architecture, New York, **(2007)**.
[14] E. Yoneki, P. Hui and J. Crowcroft, "Visualizing community detection in opportunistic networks", Proceedings of the second ACM workshop on Challenged networks, New York, **(2007)**.
[15] G. Palla, A. Barabasi and T. Vicsec, "Quatifying social group evolution", Nature, vol. 446, **(2007)**.
[16] J. M. Kumpula, M. Kivela and K. Kaski, "Sequential algorithm for fast clique percolation", Physical Review E., vol. 78, 026109, **(2008)**.
[17] K. K. Xu, J. He and S. R. Zou, "A Cohesive Subgraph Visualization based Approach to Efficiently Discover Large k-Clique Community", Arabian Journal For Science and Engineering. 10.1007/s13369-012-0299-x, **(2012)**.

[18] B. Everitt, Cluster Analysis, John Wiley, New York, **(1974)**.
[19] P. Holme, M. Huss and H. Jeong, "Subnetwork hierarchies of biochemical pathways", Preprintcond-mat/0206292, **(2002)**.
[20] E. Ravasz and A.-L. Barabasi, "Hierarchical organization in complex networks", Phys. Rev. E, 67, 026112, **(2003)**.
[21] E. Ravasz, A. L. Somera, D. A. Mongru, Z. Oltvai and A. L. Barabasi, "Hierarchical organization of modularity in metabolic networks", Science, vol. 297, **(2002)**.
[22] S. Borgatti and M. Everett, "A graph-theoretic perspective on centrality", Social Net-works, vol. 28, no. 4, **(2006)**.
[23] N. Perra and S. Fortunato, "Spectral centrality measures in complex networks", Physical review E, vol. 78, 036107, **(2008)**.
[24] H. J. Wang, J. M. Hernandez and P. V. Mieghem, "Betweenness centrality in a weighted net-work", Physical review E, vol. 77, 046105, **(2008)**.
[25] P. J. Carrington, J. Scott and S. Wasserman, "Models and methods in social network analysis", Cambridge University Press, **(2005)**.
[26] G. Palla, I. Derényi, I. Farkas and T. Vicsek, "Uncovering the overlapping community structure of complex networks in nature and society", Nature, vol. 435, **(2005)**.
[27] T. Hastie, R. Tibshirani and J. H. Friedman, "The Elements of Statistical Learning, Springer", Berlin, Germany, **(2001)**.
[28] L. C. Freeman, "Centrality in networks: conceptual clarification", Social Networks, vol. 1, **(1979)**.
[29] L. C. Freeman, "The gatekeeper, pair-dependency, and structural centrality", Quality and Quantity, vol. 14, **(1980)**.
[30] N. Perra and S. Fortunato, "Spectral centrality measures in complex networks", Physical Review E, vol. 78, 036107, **(2008)**.
[31] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases", Proceedings of the 20th International Conference on Very Large Data Bases, Santiago de Chile, Chile, **(1994)** September 12-15.
[32] X. Z. Wei, C. J. Tang and K. K. Xu, "CCDCD: Community Core Mining with Dy-namic Constrains based on Graph Density", Journal of Frontiers of Computer Science and Technology FCST, vol. 3, no. 3, **(2009)**.
[33] V. Boginski, "On structural properties of the market graph", Innovations in Financial and Economic Networks, Nagurney (editor), Edward Elgar Publishers, **(2004)**.