Proc. of the Intl. Conf. on Advances In Computer and Electronics Technology- ACET 2014.
Copyright © Institute of Research Engineers and Doctors. All rights reserved.
ISBN: 978-1-63248-024-8 doi: 10.15224/ 978-1-63248-024-8-19

An Extension of Community Extraction Algorithm on Bipartite Graph

Yanting Li, Koji Maeda, Tetsuji Kuboyama, Hiroshi Sakamoto

***Abstract*—We introduce a truss decomposition algorithm for bipartite graphs. A subgraph *G* of a graph is called *k*-truss if there are at least *k*-2 triangles containing any edge *e* of *G*. By a standard breadth-first-search algorithm, we can compute the truss decomposition for large graphs. To extract a dense substructure that represents community in graph *G*, this method avoids the intractable problem of clique detection. The truss decomposition is not, however, applicable to the bipartite graphs due to its definition. For this problem, we have proposed *quasi*-truss decomposition introducing an additional set of edges. For this decomposition, there is another problem such that dense subgraphs G_1 and G_2 are connected with a small number of edges. The previous algorithm detects the sparse structure $H = G_1 \cup G_2$ as *quasi*-truss due to the definition. In this paper, we improve the algorithm to extract denser substructures by removing such sparse edges with a top-down strategy. The extended algorithm has been implemented, and compared its performance with the previous algorithm for bipartite graphs obtained from real data.**

***Keywords*—*k*-truss, community extraction, bipartite graph.**

I. Introduction

Given a graph *G*, the communities are interpreted as cohesive subgraphs in *G*. The problem of identifying communities has attracted much attention recently due to the increased interest in studying various graphs with complicated structures [1]. It helps in analyzing graph structures, and mining useful information from graph data. Numerous techniques for data mining have been proposed for approaching graph analysis problems from different aspects [2]. Therefore, we focus on this framework of community discovery, and apply it to an attractive domain of data, such as social networks.

In this research, we consider the problem of extracting communities in a bipartite graph using the notion of *truss*, which is a substructure in a graph. Originally, the *truss* is defined as a cohesive subgraph composed of triangles, i.e., cliques with three nodes, in a graph [3], and the *truss decomposition algorithm* for extracting dense subgraphs hierarchically based on truss structure is proposed in [4].

Yanting Li, Koji Maeda, and Hiroshi Sakamoto
Kyushu Institute of Technology, Japan

Tetsuji Kuboyama
Gakushuin University, Japan

A bipartite graph is a type of the common structure for modeling relations between two classes of objects, and is found in many real-world relations such as user-item relations in an online shop. Therefore, it is an important task to extract communities from a bipartite graph by applying an efficient algorithm such as the truss decomposition. However, the truss decomposition technique is not applicable to bipartite graphs since no triangle is contained in it. To expand the notion of *truss* to the class of bipartite graph, we introduce a new notion called *quasi-truss*. We also develop an efficient algorithm for bipartite graph decomposition, and examine the scalability of it with real-world bipartite data.

We introduce an extended truss decomposition for bipartite graphs. A subgraph of a graph *G* is called *k*-truss if any edge *e* of *G* is in at least *k* - 2 triangles. By a standard breadth-first-search algorithm, we can compute the truss decomposition for large graphs. When extracting a dense substructure community, this method avoids the intractable problem of clique detection. This simplicity is an advantage. The truss decomposition is not, however, applicable to the bipartite graphs due to the definition. For this problem, we have proposed quasi-truss decomposition of a bipartite graph *H*, i.e., *H* is transformed into *H'* by adding the special edge (*x*, *y*) if there are two edges (*x*, *z*) and (*y*, *z*) in *H*, and the quasi-truss of *H* is obtained by removing all special edges from the truss of *H'*. For this decomposition, consider the case that two dense subgraphs G_1 and G_2 are connected with a small number of edges. Then, the previous algorithm in [5] detects the structure $H = G_1 \cup G_2$ as quasi-truss due to the definition. In this paper, we improve the algorithm to extract more dense substructures by removing such sparse edges between dense graphs. We implemented the extended algorithm and compare its performance with the previous algorithm for bipartite graphs defined on real data.

II. Related Works

An interesting substructure in a graph is called *community* which is a subgraph densely connected by edges among nodes. According to the definition by Flake et al. [6], a community is a set of nodes in which each member has at least as many edges connecting to members as it does to non-members. This definition is unambiguous, and for any set of nodes, we can determine whether it is a community or not.

In [7,8], a community of a graph *G* = (*V*, *E*) is defined as a subgraph containing at least one clique, i.e., a subset $V' \subseteq V$ such that the subgraph in *G* induced by *V'* is a complete graph. Generally, the clique is extracted as a set of the nodes with high degrees. For this reason, the nodes with relatively lower

Proc. of the Intl. Conf. on Advances In Computer and Electronics Technology- ACET 2014.
Copyright © Institute of Research Engineers and Doctors. All rights reserved.
ISBN: 978-1-63248-024-8 doi: 10.15224/ 978-1-63248-024-8-19

degrees are liable to be ignored, and not so much effective for uniformly sparse graphs. Moreover, the problem of finding maximal cliques is computationally hard. Thus, in the last decade, several efficient algorithms to find *quasi-cliques*, instead of extract cliques, have been proposed [9].

The quasi-clique is a relaxation notion of clique, for example, on the density [10] or the degree [11, 12]. However, the problem of finding these quasi-cliques remains to be NP-hard. Moreover, it may be difficult to capture the entire structure of communities in a graph since these subgraphs may substantially overlap, or completely be separated.

To address these difficulties, a definition of dense subgraph called k-core has been proposed. It is defined as a maximal connected subgraph among all of its nodes with higher degree than k in G. Besides, the truss decomposition algorithm has been proposed: given a graph G, the k-truss of G is the largest subgraph of G in which any edge is contained in at least k - 2 triangles within the subgraph [13]. The problem of truss decomposition is to find all k-trusses where $k \geq 3$.

While the problem of finding the densest subgraph is NP-hard, there is an efficient polynomial algorithm for the k-truss detection. From the point of view of the clique approximation, the k-truss is better than k-core [14,15], which is a well-known subgraph for community discovery. For the problem of finding all k-trusses in a graph, i.e., truss decomposition problem, an efficient in-memory algorithm [3] and two I/O-efficient algorithms have been presented to handle massive networks [4], and the efficiency of truss decomposition has been proved.

Many interesting relations are represented by bipartite graphs, such as user-item relations in an online shop, the user-music/movie relations of an online entertainment system. Recently, we have proposed an algorithm for enumerating triangles in a bipartite graph [5]. In this paper, we improved it, and propose a new quasi-truss decomposition algorithm. Our algorithm is based on the following fundamental algorithms for bipartite graphs.

One is for testing bipartiteness to examine whether a graph is a bipartite or not [16]. The main idea of testing bipartiteness algorithm is to assign every node with a certain color in order to distinguish the color of its parent in a preorder traverse. This provides a two-colored spanning tree which consists of the edges containing nodes to their parents. However, some nodes may not be colored properly. In the case of depth-first-search, one of the two endpoints of every non-tree edge is another endpoint's ancestor. These pairs of nodes have different colors when non-tree edges are found. An odd-cycle can be formed by the path from ancestor to descendant within the incorrect colored edges together. With such an evidence, the graph is not bipartite. Every edge should be colored properly if the algorithm is terminated without detecting any odd-cycle of this type. It returns a bipartite graph with colors.

Another one is the matching algorithm on bipartite graph. Matching in a graph $G = (V, E)$ is a subset of E such that no two edges share a common node. A node is matched if it is an endpoint of one of the edges in the matching. Matching problem is easier to solve by using bipartite graph than non-bipartite graph in many cases, such as the popular Hopcroft-Karp algorithm [17] for maximum cardinality matching which working correctly only with bipartite graphs.

III. Basic Notion

A clique consisting of three nodes is called a triangle. A *truss* is defined based on triangles embedded in a graph. For a threshold k, the k-truss is a type of cohesive subgraphs that represents the largest subgraph of G such that every edge is contained in at least k - 2 triangles within the subgraph. This value is called the support of edge $e = (u, v) \in E(G)$, denoted by $sup(e, G)$. When G is unnecessary to indicate, we denote just $sup(e)$. The support of e in G is the number of triangles in G that contain e. Thus, the k-truss for $k \geq 2$, denoted T_k, is a subgraph defined by the condition $\forall e \in E(T_k)\ [sup(e, T_k) \geq k - 2]$.

The task of truss decomposition is to find all k-truss in G for $k \geq 2$ where k is the bounded by the degree of G. The truss number of an edge e in G is defined as $\max\{k : e \in E(T_k)\}$, denoted by $\Psi(e)$. From the definition of truss number, another definition k-class that denoted by Φ_k, defined as $\{e : e \in E(G), \Psi(e) = k\}$. Relatively, the k-truss is the edge induced subgraph from the set $E(T_k) = \cup_{i \geq k} \Phi_i$.

The Figure 1 Illustrates the k-truss decomposition of a given graph G. The edges are contained in different number of triangles in G. The 2-class Φ_2 is the set of edges e with $sup(e) = 0$. The 3-class Φ_3 is the set of edges with $sup(e) = 1$, i.e., for $e = (x, y)$, there exist at least one node z such that $(x, z), (y, z) \in E(G)$. The 4-class is analogous.

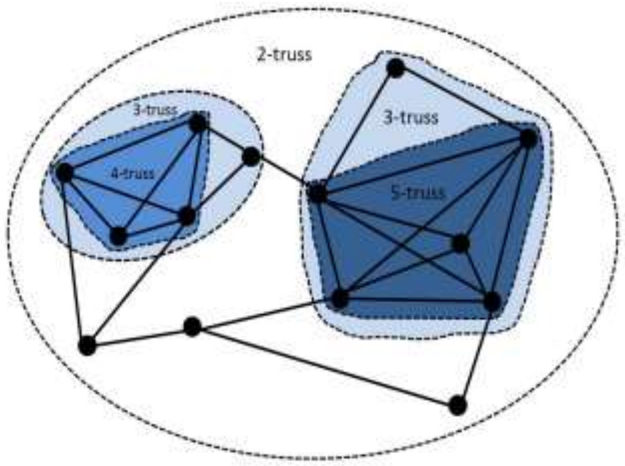

Figure 1. Illustration of the 2-, 3-, 4-, and 5-truss decomposition

From the k-classes, k-trusses of G can be obtained as follow. The 2-truss T_2 is simply G itself. The 3-truss T_3 is the subgraph formed by the edge set $\Phi_2 \cup \Phi_3 \cup \Phi_4$, etc. It can be verified that each edge of T_k is contained in at least k - 2 triangles where $2 \leq k \leq 5$. The k-trusses represent the hierarchical structures of G at different level of granularity.

Proc. of the Intl. Conf. on Advances In Computer and Electronics Technology- ACET 2014.
Copyright © Institute of Research Engineers and Doctors. All rights reserved.
ISBN: 978-1-63248-024-8 doi: 10.15224/ 978-1-63248-024-8-19

IV. Quasi-truss Decomposition and Its Improvement

A. *Quasi-truss for Bipartite Graph*

Given a bipartite graph $G = (V_1 \cup V_2, E)$, there is no edge connecting all nodes in V_1 or V_2. Obviously, a bipartite graph contains no triangles due to its characteristic. Thus, we extend the notion of *k*-truss to bipartite graph, named *quasi-k-truss*. Based on the definition of *k*-truss, we introduce a special edge $e' \in E'$ to the bipartite graph G as follow. For any two distinct nodes u and v in the same node set, if at least one common neighbor of them in another node set, define the special edge $(u, v) \in E'$. The bipartite graph G is transformed to $G' = (V_1 \cup V_2, E \cup E')$ such that $E' = \{(u, v) \mid u, v \in V_1$ or $u, v \in V_2$, and a node in $V(G)$ is adjacent to $u, v\}$. The substructures of G can be obtained by recursively removing the edges in which are contained in the triangles.

Conceptually, the notion of quasi-*k*-truss is similar to *k*-truss. To compute the quasi-*k*-truss, initially, every node in both V_1 and V_2 of the given bipartite graph G will be visited. The set of special edges is determined by checking whether any two adjacent nodes in the same node set connected by a special edge share at least one common neighbor node in another node set. The structure of the given bipartite graph and its special edges is illustrated in Figure 2.

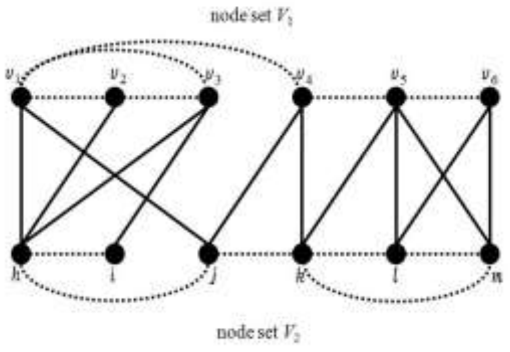

Figure 2. The generation of special edge in bipartite graph

As shown in Figure 2, there are two types of edges in the bipartite graph: the original edges in E are depicted by solid lines, and the special edges in E' exists between two nodes v_1 and v_2 in V_1, provided that v_1 and v_2 share a common neighbor node $v_h \in V_2$. Then, an imaginary triangle is formed by three edges $\{(v_1, v_2), (v_2, v_h), (v_1, v_h)\}$. By this preprocessing, a bipartite graph G is transformed into G'. The quasi-*k*-truss of G is a subset of $G(E)$ obtained by removing all special edges from the *k*-truss of G'.

The truss-like components in each hierarchy is illustrated by Figure 3. According to the definition of quasi-*k*-truss, the truss-like components consists of edges $e \in E$. The special edges $e' \in E'$ is excluded. Initially, the 2-truss is the bipartite graph itself. The 3-truss contains all edges anchored in no less than a triangle. The 4-truss contains all edges anchored in no less than two triangles. It contains edges $\{(v_1, h), (v_2, h), (v_3, h), (v_4, j), (v_4, k), (v_5, k), (v_5, l), (v_5, m), (v_6, l), (v_6, m)\}$. The 5-truss contains edges $\{(v_1, h), (v_5, k), (v_5, l), (v_5, m)\}$. The given bipartite graph is decomposed hierarchically.

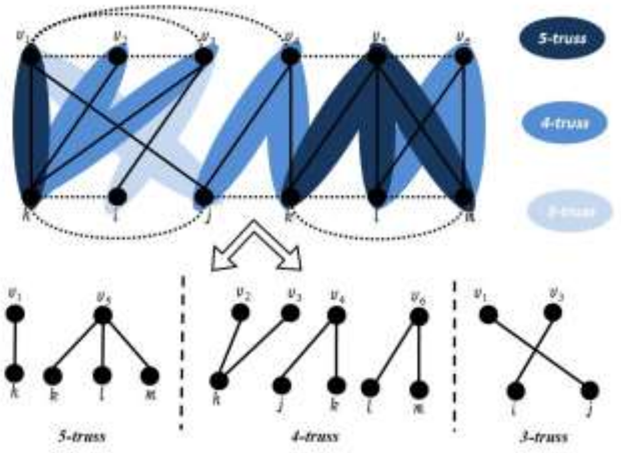

Figure 3. The quasi-*k*-truss decomposition in bipartite graph, where almost special edges are omitted due to the complication

B. *Improvement*

The algorithm for extracting quasi-*k*-truss have been proposed in [3]. However, this algorithm contains a drawback when the input bipartite graph is dense. Consider the situation that two complete bipartite graphs G_1 and G_2 that they are connected by an edge, shown as Figure 4.

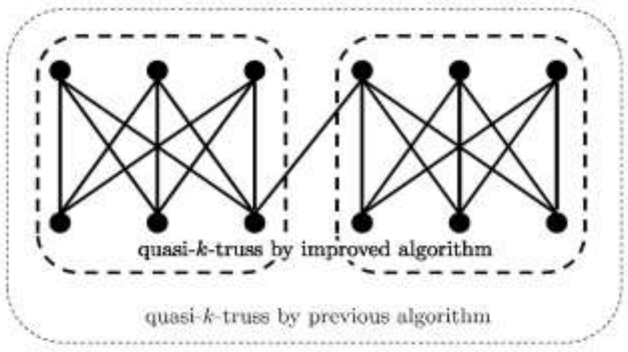
quasi-k-truss by improved algorithm
quasi-k-truss by previous algorithm

Figure 4. An example of quasi-k-truss detected by the improved algorithm

The previous algorithm extracts the structure $E(G_1) \cup E(G_2) \cup \{e\}$ because e is detected as an edge of higher quasi-*k*-truss than other edge in G_1 or G_2. The reason is that e is most frequent edge appearing in triangles. To avoid such case, we develop an improved algorithm described in *Algorithm 1*. In this algorithm, we propose a method for removing those undesired edge e based on the frequency of triangles contain e.

Proc. of the Intl. Conf. on Advances In Computer and Electronics Technology- ACET 2014.
Copyright © Institute of Research Engineers and Doctors. All rights reserved.
ISBN: 978-1-63248-024-8 doi: 10.15224/ 978-1-63248-024-8-19

***Algorithm 1**: Improved truss-decomposition algorithm*

```
• input graph $G$ is transformed into $G'$
• $T$: the set of all triangles in $G'$
• $q$: the queue for graph traverse
• $e$: an edge in $E = E(G)$
• $e'$: a special edge in $E' = E(G')$
• $Q$: maximum number of triangles with a single edge in $E'$
• $C_{(e)}$: number of adjacency edges of an edge $e$
Input: $G = (V_1 \cup V_2, E)$
Output: all detected subgraphs
1.  $T := \varnothing$
2.  for all $v \in (V_1 \cup V_2)$ do
3.      $q.enqueue(v)$
4.      while not $q.empty()$ do
5.          $v := q.dequeue()$
6.          if $(v_1, v_j)$ , $(v_1, v_k) \in E(G)$ then
7.              $e' := (v_j, v_k)$ generated
8.              $T = T \cup \{t\}, t=\{v_1, v_j, v_k\}$
9.          end
10.         transform $G$ into $G' = (V_1 \cup V_2, E \cup E')$
11.     end
12. end
13. for all $e \in G'$ do
14.     if $e$ is contained in at least $Q$ triangles then
15.         remove $e$ from $G'$
16.     end
17.     for all edges adjacent to $e$ do
18.         $C_{(e)} := C_{(e)} - 1$
19.     end
20.     if $C_{(e)} < Q$ then
21.         $Q := Q - 1$
22.     end
23.     output all edges in subgraphs of $G$
24.   end
25.   return
26. end
```

In *Algorithm 1*, the triangle set T of G is initialized as an empty set before the process begins. The first loop processing from step 2 to step 12 is the same as the previous proposed *truss decomposition algorithm* in [5]. All nodes of G is visited once by adopting the standard breadth-first-search algorithm. Then, the special edge $e' \in E'$ is generated for constructing triangles in G. So the given bipartite graph G is transformed into G', where $G' = (V_1 \cup V_2, E \cup E')$. In the next loop processing, from step 13 to step 26, all of the edge $e \in E$ is traversed, but excluding the special edges $e' \in E'$ because the special edge $e' \in E'$ is used for constructing triangles in the given bipartite graph G. The edge $e \in E$ are removed from the graph G recursively based on the number of triangles which containing the edge e. However, the number of triangles in which containing the edge e and its adjacency edges decreases. In here, a novel definition of edge $e \in$ E is proposed. It is the number of edges in which adjacent to the edge e, denoted by $C_{(e)}$ in the *algorithm 1*. For example, the triangle constructed by the nodes v, v_j, v_k is deleted if the edge (v, v_k) or (v, v_j) is removed from G. Then, the value of $C_{(e)}$ of edge (v, v_k) or (v, v_j) is $C_{(e)} - 1$. This process is proposed for preventing the drawback of edges' multi traversal, and removing the sparse edges e from G. The value of threshold Q decreases when the value $C_{(e)}$ of any edge e is smaller than the threshold Q. Finally, the set of edges that contained in the dense subgraphs of G is extracted as the output result.

V. Experimental Results

A succession of experiments has been done to observe the density and size of the extracted substructures. The results verify the effectiveness of the improved algorithm for the quasi-*k*-truss decomposition of bipartite graphs. The environment is CPU: Intel core i7 2.3GHz, RAM: 8GB, and the version 4.1.2 of C/C++ compiler in Mac OS 10.8.3.

The dataset is chosen from 20 newsgroups, which were referred in [16]. They were a collection of newsgroup documents. Each of them is corresponding to a certain topic, and represents the relationship between keywords and news documents. The bipartite graph G constructed from the dataset is characterized by $|V(G)| = 444$, $|E(G)| = 578$, and $degree(G) = 56$.

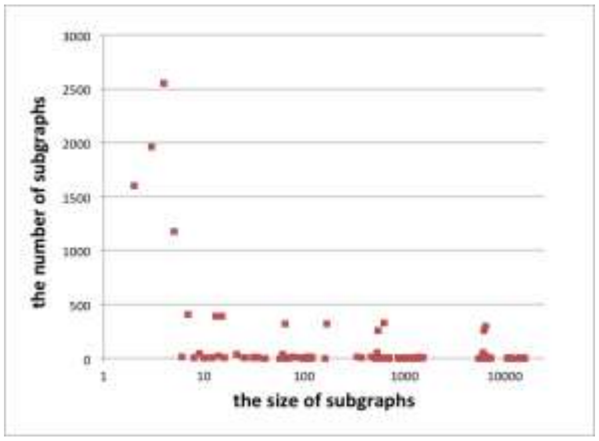

Figure 5. The result by the improved algorithm (#subgraphs)

The Figure 5 shows the number of subgraphs extracted by the improved algorithm. The *X*-axis denotes the size of subgraphs, and the *Y*-axis is the number of extracted subgraphs. Compared to Figure 5, the Figure 6 shows the number of subgraphs extracted by the previous algorithm. By these results, we conclude the efficiency of our algorithm in the view point of the number of extracted substructures.

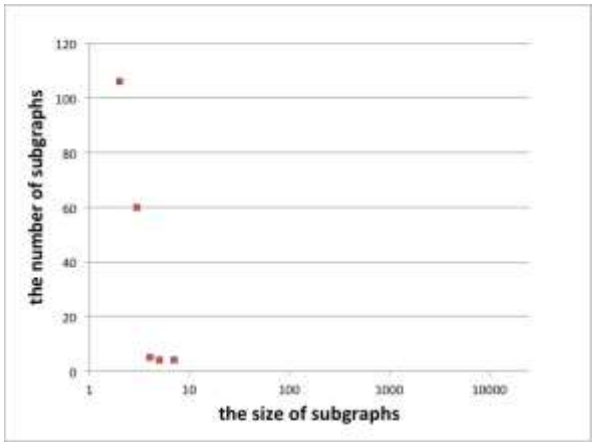

Figure 6. The result by the previous algorithm (#subgraphs)

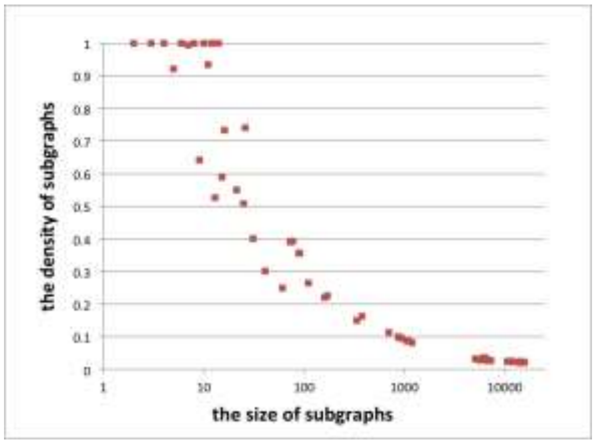

Figure 7. The result by the improved algorithm (density)

The Figure 7 Shows the average of density of extracted subgraphs by the improved algorithm. The *X*-axis is the size of extracted subgraphs formed by the node sets V_1 and V_2, and *Y*-axis is its density, i.e., the ratio $|E(G)|/(|V_1||V_2|)$. On the other hand, the previous algorithm extracts almost trivial bipartite graph so that $|V_1| = 1$ or $|V_2| = 1$. By this result, out algorithm can extracts sufficiently dense substructures from the given bipartite graph *G*.

VI. Conclusion

A novel quasi-*k*-truss decomposition algorithm has been proposed for bipartite graphs. This is an improvement of the previous version of quasi-*k*-truss. As the experimental results show, the new algorithm works well compared to the previous algorithm. The scalability is a future work of the proposed algorithm since the number of special edges grows rapidly when the input graph is dense. For this problem, a method for pruning of special edges is needed. More experiments will be done with more data sets to evaluate the effectiveness and efficiency of the proposed algorithm.

Acknowledges

This work was supported in part by JSPS KAKENHI Grant Numbers 26280088 and 26280090.

References

[1] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, & D. Parisi, Defining and identifying communities in networks. Proceedings of the National Academy of Sciences of the United States of America, 101(9), 2658-2663, 2000.

[2] G.W. Flake, S. Lawrence, C.L. Giles & F.M. Coetzee, Self-organization and identification of web communities. Computer, 35(3), 66-70, 2002.

[3] J. Cohen, “Truss: cohesive subgraphs for social network analysis,” 2008.

[4] J. Wang, J. Cheng, Truss Decomposition in Massive Network, VLDB2012, 2012, pp.812–823.

[5] Y. Li, T. Kuboyama, H. Sakamoto, Mining Twitter Data: Discover Quasi-truss from Bipartite Graph, in Magnetism, KES-IDT, 2103, pp. 287–295.

[6] G. W. Flake, S. Lawrence, C. L. Giles, Efficient Identification of Web Community, KDD2000, 2000, pp.150-160.

[7] J. M. Kleinberg, Authoritative Sources in a Hyperlinked Environment, SODA1998, 1998, pp.668-677.

[8] R. Kumar, P. Raghavan, S. Rajagopalan, A. Tomkins, Extracting Large-scale Knowledge bases from the Web, VLDB1999, 1999, pp. 639–650.

[9] J. Cheng, Y. Ke, A.W.C. Fu, J.X. Yu & L. Zhu, Finding maximal cliques in massive networks by h*-graph. In Proceedings of the 2010 ACM SIGMOD International Conference on Management of data (pp. 447-458). ACM.

[10] J. Abello, M.G. Resende & S. Sudarsky, Massive quasi-clique detection. In LATIN 2002: Theoretical Informatics (pp. 598-612). Springer Berlin Heidelberg.

[11] H. Matsuda, T. Ishihara, A. Hashimoto, Classifying Molecular Sequences using Lingkage Graph with their Pairwise Similarities, Theor. Compt. Sci., 1999, 201(2)305-325.

[12] J. Pei, D. Jiang, A. Zhang, On Mining Cross-graph Quasi-cliques, SIGKDD2005, 2005.

[13] J. Cohen, Graph Twiddling in a Mapreduce World, Computing in Science and Engineering, 2009, 11(4)29-41.

[14] V. Batagelj, M. Zaversnik, An O(n) Algorithm for Cores Decomposition of Networks, advances in data analysis and classification, 2011, Vol. 5, No. 2, pp. 129-145.

[15] S. B. Seidman, Network Structure and Minimum Degree, Social Networks, 1983, 5(3)269-287.

[16] N. Alon, M. Krivelevich, Testing k-colorability, SIAM J. Discrete Math., 2002, 15(2)211-227.

[17] J. E. Hopcroft and R. M. Karp, An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs, SIAM J. Comput., 1973, 2(4)225-231.

[18] J. Chen, Y. Saad, Dense Subgraph Extraction with Application to Community Detection, Knowledge and Data Engineering, IEEE Trans 2012, Vol. 24, Issue: 7.

[19] J. Cheng, Y. Ke, A. W. C. Fu, J. X. Yu, L. Zhu, Finding Maximal Cliques in Massive Networks, ACM Transactions on Database Systems, 2011, 36(4) Article, No. 21.

[20] L. Danon, A. Diaz-Guilera, J. Duch, A. Arenas, Comparing Community Structure Identification, Journal of Statistical Mechanics: Theory and Experiment, 2005, Vol. 2005, p. p09008.