Truss Decomposition for Extracting Communities in Bipartite Graph

Yanting Li
Graduate School of Computer Science
and Systems Engineering
Kyushu Institute of Technology
Iizuka, Japan
Email: k791502g@ai.kyutech.ac.jp

Tetsuji Kuboyama
Computer Centre
Gakushuin University
Tokyo, Japan
Email: ori-immm2013@tk.cc.gakushuin.ac.jp

Hiroshi Sakamoto
Graduate School of Computer Science
and Systems Engineering
Kyushu Institute of Technology
Iizuka, Japan
Email:hiroshi@ai.kyutech.ac.jp

***Abstract*—We propose a novel method for extracting communities, i.e., dense subgraphs, embedded into a bipartite graph. Our method is based on a technique for graph decomposition. Decomposing a large graph into cohesive subgraphs plays an important role in identifying community structures in social network analysis. Among a lot of definitions of cohesive subgraphs, the *k*-truss formed by triangles is one of the simplest cohesive subgraphs with a good trade-off between computational efficiency and clique approximation. This decomposition is, however, not applicable to bipartite graphs because bipartite graphs contain no triangles. In this paper, a *quasi*-truss decomposition algorithm for bipartite graphs is proposed based on the truss decomposition algorithm for general graphs. The proposed method can be used for analyzing the international business, such as the relationship between clients and sales volume in a certain period, and also analyze the social networking, such as users-topics relations in the twitter community.**

***Keywords*—bipartite graph, triangle, truss decomposition, dense subgraph, community discovery.**

I. Introduction

Communities are interpreted as dense subgraphs in a given graph G. The problem of identifying communities has attracted much attention recently due to the increased interest in studying various graphs with complicated structures. It helps to analyze graph structures, and mining useful information from graphs. Various techniques of data mining have been proposed for approaching graph analysis problems from different aspects. 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 dense structure in a graph. Originally, the *truss* is defined as a dense subgraph composed of triangles, i.e., cliques with three nodes, in a graph [1], and the *truss decomposition* algorithm for extracting hierarchical dense subgraphs based on truss structures is proposed [2].

On the other hand, a bipartite graph is a common structure for modeling relations between two classes of objects, and is found in many real-world data sets such as user-item relations in an online shop. The truss decomposition is not applicable to bipartite graphs since any bipartite graphs include no triangles. To expand the notion of *truss* to the class of bipartite graphs, 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.

Organization: Section II introduces some related works about community extraction and bipartite graph analysis. Section III introduces basic notions used in this paper. In Section IV, we propose the *quasi*-truss decomposition algorithm. The experiments verify the efficiency of this algorithm for graph analysis in Section V. Finally, Section VI concludes the paper.

II. Related Work

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. [4], 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 [5], [6], 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 degrees are liable to be ignored, and are 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 exact cliques, have been proposed.

The *quasi*-clique is a relaxation notion of clique, for example, on the density [7] or the degree [8], [9]. However, the problem of finding these *quasi*-cliques remains NP-hard. Moreover, it may be difficult to capture the entire structure of communities in a graph since these subgraphs may substantially overlap, or be 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 [10]. The problem of truss decomposition is to find all k-trusses where $k \geq 3$.

 ISBN: 978-1-61208-311-7

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* [11], [12], 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 [1] and two I/O-efficient algorithms [2] have been presented to handle massive networks, 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. Recently, we have proposed an algorithm for enumerating triangles in a bipartite graph [3]. In this paper, we improve 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 [13]. 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 connecting nodes to their parents. However, some nodes may be not 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 color.

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 [14] for maximum cardinality matching which working correctly only with bipartite graphs.

III. Basic Notion

A triangle is one of the fundamental structures of graph that represents the smallest non-trivial clique. Indeed, the triangle plays an important role in graph analysis, especially in the computation of clustering coefficient, the triangular connectivity and transitivity ratio in massive networks. Three nodes in a triangle are fully connected by three edges formed by nodes $\{v_1, v_2, v_3\}$ that either directed edge or undirected edge, denoted as follows:

$$T_{123} = \{(v_1, v_2), (v_2, v_3), (v_1, v_3)\}$$

The notion of truss is defined by such triangles embedded in a graph. For the 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 an edge $e = (u, v) \in E_G$, denoted by $sup(e)$. The support of an edge e in G is the number of triangles in G that contain e. Thus, the k-truss of G where $k \geq 2$, denoted as T_k so that $\forall e \in E_{T_k}$, $sup(e, T_k) \geq (k-2)$. The task of truss decomposition in G is to find all trusses in G where $2 \leq k \leq k_{max}$. The k_{max} denotes the maximum truss number of any edge in G. The truss number of an edge e in G is defined as $max\{k : e \in E_{T_k}\}$, denoted by $\phi(e)$. From the definition of truss number, another definition k-class that denoted by Φ_k, defined as $\{e : e \in E_G, \phi(e) = k\}$. Relatively, the k-truss can be obtained from the set of edges $E_{T_k} = \cup_{i \geq k} \Phi_i$.

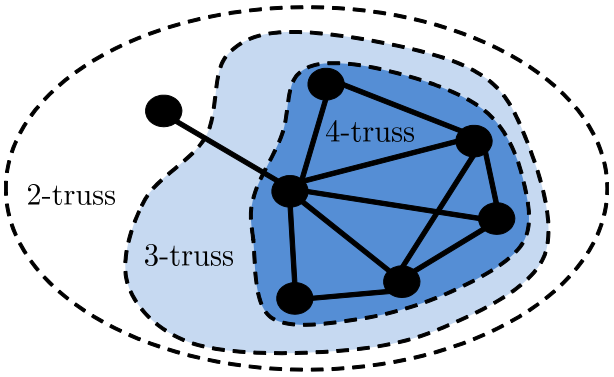

Fig. 1. Illustration of the 2-, 3-, and 4-truss decomposition

Fig. 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.

From the k-classes, k-trusses of G can be obtained as follows. The 2-truss T_2 is simply G itself. The 3-truss T_3 is the subgraph formed by the edge set $\Phi_3 \cup \Phi_4 \cup \Phi_5$, etc. It can be verified that each edge of T_k is contained in at least $k-2$ triangles for $2 \leq k \leq 5$. The k-trusses represent the hierarchical structures of G at different level of granularity.

On the other hand, there are many relations represented by bipartite graphs, which are equivalent to transaction data. However, as shown in Fig. 2, bipartite graph contains no triangle due to the definition: the node set is divided into two disjoint subsets V_1, V_2 such that no edge (u, v) $(u \in V_1, v \in V_2)$ is defined. Thus, we propose an extended version of the truss decomposition for bipartite graphs in the following section. Now, we prepare some important notions in our algorithm.

Given a bipartite graph $G = (V_1 \cup V_2, E)$, the algorithm transforms G to $G' = (V_1 \cup V_2, E \cup E')$ such that $E' = \{(u, v) \mid u, v \in V_1, u \neq v$, and $x \in V_2$ is adjacent to $u, v\}$.

We call $e' \in E'$ the *special edge*. For two distinct adjacent edges $e_1, e_2 \in E$ in G, there exists a triangle with a special edge $e' \in E'$ in G'. With more triangles sharing a unique special edge e' in G', a dense subgraph in G is expected to be identified. We introduce a novel notion of dense subgraph in bipartite graphs, the *quasi*-truss. The *quasi*-truss of G' is defined as the largest subgraph in G' containing exactly one special edge e'. Consequently, we obtain the substructure of G by removing all $e' \in E'$ from the *quasi*-truss.

In the following section, we design an algorithm to extract such components from a large network data.

 ISBN: 978-1-61208-311-7

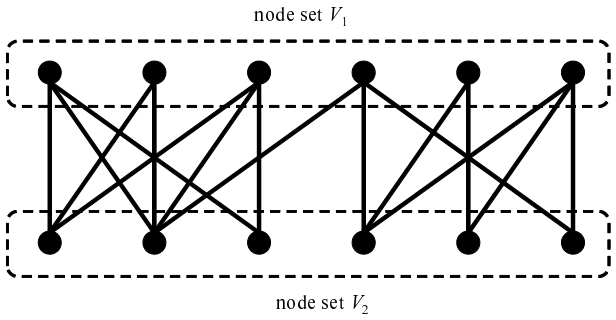

Fig. 2. The structure of a bipartite graph

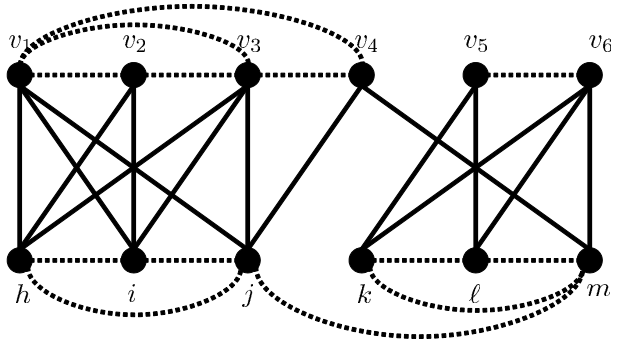

Fig. 3. Generate edges in a bipartite graph

IV. Quasi-Truss Decomposition

A. Quasi-truss

Conceptually, the definition of *quasi*-truss is similar with *k*-truss. In a given bipartite graph G, clearly, G contains no triangle. Then we define special edges $e' \in E'$ among nodes included in V_1 or V_2 exclusively. Initially, every node in both two node sets of the bipartite graph G will be visited. Determine whether any two adjacent nodes in the same node set sharing one common neighbor node in another node set or not. Then, the connectivity occurs between these two nodes, and connected by a special edge, denoted as e' if these two nodes have one common neighbor node in another node set of G. More formally, a special edge $e' = (x,y)$ is defined for $x,y \in V_1$ if there exists $z \in V_2$ such that $(x,z),(y,z) \in E$. After generating all special edges e', the structure of original bipartite graph is transformed to $G' = (V_1 \cup V_2, E \cup E')$.

In this definition, an edge $e' = (v_i, v_k)$ is generated in G so that $v_i, v_k \in V_1$ have a common neighbor node $v_j \in V_2$. The edge e' is denoted as e'_{ik} where $i,k \in V_1$ and $e'_{ik} \notin E$. The special edge e' is essential to form a triangle in bipartite graph G. The number of common neighbor nodes of edge e' is simply equal to the number of triangles which contain the edge e'. The common neighbor nodes of e' belong to the node set which does not contain two endpoints of e'. All of the triangles T belong to the bipartite graph G.

According to the definition, the Q-truss is the trusses of reconstructed G' where $Q > 0$. Here, the Q indicates the hierarchy of *quasi*-truss in order to distinguish from *k*-truss. Thus, the maximum *quasi*-truss of G can be defined as the special edge e' contained in maximal number of triangles in G.

When $Q = 1$, the *quasi*-truss is simply G itself since one edge e' is contained in one triangle exactly. We suppose that e'_{ik} has only one neighbor node v_i in another node set. Then, three nodes form a triangle $T_{ijk} = \{(v_i, v_j),(v_j, v_k),(v_i, v_k)\}$ which is a 1-truss subgraph for e'_{ik} contained in one triangle exactly. This differs from the definition of *k*-truss decomposition algorithm.

Fig. 3 illustrates the generation of edge e' in a given bipartite graph G. There are two types of edges: the original existed edges $e \in E$ which is in solid line, and the generated edge $e' \in E'$ which are illustrated by the dotted line. For instance, two nodes v_2 and v_3 that are in the same node set have a common neighbor node i, then, v_2 and v_3 will be connected by one generated edge e' so that a triangle $T_{23i} = \{(2,3),(3,i),(2,i)\}$ is formed. At the same time, the edge e'_{23} also contained in another triangle in G, the triangle $T_{23h} = \{(1,2),(2,h),(1,h)\}$ for node v_h also their common neighbor node. Thus, the four nodes v_2, v_3, v_h and v_i can be considered as 2-trusses for both two generated edges e'_{23} and e'_{hi} are contained in two triangles. In another situation, the nodes v_1 and v_2 have only one common neighbor node h that the generated edge e'_{12} is contained in only one triangle. The subgraph that contains three nodes v_1, v_2 and v_h can be considered as 1-truss. The nodes in the same node set such as the nodes v_4 and v_5 do not connected by any edge e' as they do not have any common neighbor node in another node set.

B. Decomposition algorithm

Quasi-truss decomposition algorithm is summarized in algorithm 1.

We employ the hash table to store and sort the special edge e' in this improved *quasi-truss decomposition algorithm*. Initialize the hash table of E', denoted as $hash[E']$, and the triangle set, denoted as T. The graph traverse begins from node v. A special edge e'_{jk} is generated to connect any two nodes v_j and v_k directly connect to v. Then, $T = \{v, v_j, v_k\}$ can be formed in G after this process. An e' is contained in at least one triangle where $Q = 1$. All of $e' \in E'$ is stored in the $hash[E']$, and sorted hierarchically. A common neighbor node of an e' represents a vertex of a triangle. For any two nodes in the same node set connected by an e', the number of their common neighbor nodes is equivalent to the number of triangles contains the e'. The $hash[E']$ only stores each unique edge e' instead of storing all triangles in an array [3]. The memory usage can be significantly reduced. Moreover, pointer is adopted to point to the common neighbor nodes of each e'. To extract the maximum *quasi*-truss represents the largest community, it was essential that counted the total number of common neighbor nodes of each e' in $hash[E']$, and output the e' with maximal number of common neighbor nodes. Next, the e' in $hash[E']$ is removed iteratively based on the number of its common neighbor nodes. For example, an e' will be removed from $hash[E']$ if the e' has less than five common neighbor nodes, or in other words, an e' is contained in less than five triangles where $Q = 5$. Finally, enumerate all triangles which satisfy the parameter. These enumerated triangles represent the dense subgraphs of G in different hierarchy.

 ISBN: 978-1-61208-311-7

Algorithm 1 *Quasi-Truss Decomposition Algorithm*

- q := queue for graph traverse
- Q := input threshold of hierarchy of trusses
- E' := the edge set contains all special edge e'
- $hash[E']$:= the hash table of E'
- T := a set of triangles in G
- $num_{(v)}$:= number of nodes belong to an e'

Require: $G = (V_1 \cup V_2, E)$, Q = 1,2,3,4....m
Ensure: T within Q hierarchy

```
1: init hash[E'] = φ, T = φ;
2: for all v ∈ (V1 ∪ V2) do
3:    v.mark = 0
4:    q.enqueue(v0);
5:    while not q.empty() do
6:       v = q.dequeue()
7:       v.mark = 1;
8:       if v ∈ e(v, vj) ∩ e(v, vk) then
9:          e' = (vj, vk) generated;
10:         hash[E'] = hash[E'] ∪ hash[e'jk]
11:      end if
12:      T = T ∪ {v, vj, vk}
13:   end while
14:   for Q = 1 to m do
15:      for all e' ∈ hash[E'] do
16:         if num(v) ∈ e' < Q then
17:            remove e' from hash[E']
18:         end if
19:      end for
20:   end for
21: end for
22: output T contains e' within Q hierarchy
```

V. Experiment and Evaluation

We observed the performance of the proposed method via a succession of experiments in this section. The experimental results evaluated the effectiveness of the *quasi*-truss algorithm. All of the experiments were done on a machine with the Inter i7 2.3GHz CPU, 8GB RAM, and the version 4.1.2 of C compiler in Mac OS 10.8.3.

A. Data characteristics

Five real-world graph datasets with different sizes were used in these experiments. Table I indicates the features of the datasets. $|V_1|$ and $|V_2|$ indicated the number of nodes of each node-set in these given bipartite graphs. $|E|$ showed the number of edges in each dataset.

TABLE I. Features of datasets

File name	$\|V_1\|$	$\|V_2\|$	$\|E\|$	size(kb)
cmuDiff	3,000	5,932	263,325	32.1
cmuSame	3,000	7,666	185,680	46.6
cmuSim	3,000	10,083	288,989	260
HetRec	9,372	6,257	26,232	259
MovieLens	3,706	6,040	1,000,209	40.1

The three datasets *cmuDiff*, *cmuSame* and *cmuSim* were chosen from 20 newsgroups datasets, which were also refered in [16]. They were collections of newsgroup documents. Each of them corresponded to a certain topic, and recorded the relationship between keywords and news documents. Both *HetRec* and *MovieLens* were two datasets released from the framework of Information Heterogeneity and Fusion in Recommender Systems. The *HetRec* recorded the relationship between online users and artists/musics from *Last.fm online music system* in 2011. The *MovieLens* dataset contained anonymous ratings by MovieLens users toward a certain number of movies in 2000.

Matrix blocking proposed in [16] was a community detection technique based on the connectivity occurence among all nodes in G. Oppositely, the proposed algorithm in this paper was designed to decompose a bipartite graph, and identify the subgraphs within different hierarchy. Therefore, in these experiments, we mainly observed three aspects for evaluating the proposed algorithm. First, we stated the total number of triangles which included all special edges e'. Second, we observed the time cost for enumerating all triangles in each bipartite graph. Finally, the largest community structure represented by the maximum *quasi*-truss was extracted from each bipartite graph.

B. Experimental results

Table II shows the experimental results by using the five datasets. The #T indicates the total number of triangles formed in each given bipartite graph. The next column shows the time cost for triangles' forming. The results of Q_{max} clearly indicates the maximal *quasi*-truss in each bipartite graph. The maximal *quasi*-truss represents the largest communities in each given bipartite graph.

TABLE II. Statistics of experimental results

File name	#T	size(kb)	time(in sec.)	Q_{max}
cmuDiff	61,638	874	0.374	238
cmuSame	174,363	2,458	0.78	390
cmuSim	1,838,827	27,471	7.207	112
HetRac	945,043	15,020	6.739	351
MovieLens	63,271	891	0.453	223

The running time increased linearly with the increasing number of triangles. Meanwhile, the number of edges $e' \in E'$ also increased. But it was worth to notice that the number of edges $e' \in E'$ did not equal to the total number of triangles for an edge e' was contained in more than one triangles.

In the third column of Table II, we observed the maximal *quasi*-truss for each given dataset. According to the definition of *quasi*-truss, the subgraph with Q_{max} represented the largest community in which a unique edge e' was contained in maximal number of triangles. Thus, the subgraph with Q_{max} was the densest subgraph, since it represented the core of a given graph. In this experiment, "Q_{max}"-truss were extracted from the given G by adopting a technique which was similar to the *Top-Down approach of truss decomposition* concluded in [2]. Thought observing the experiment results, the maximum *quasi*-truss of G also increased with the total number of triangles. Meanwhile, the value of "Q_{max}" was difficult to estimate. However, the result of dataset *cmuSim* was an exception although this dataset contained the maximal number of triangles compared with results of other datasets. The value of Q_{max} of *cmuSim* dataset was the smallest. This result illustrated

 ISBN: 978-1-61208-311-7

that the connectivity among all nodes in both V_1 and V_2 of G had a significant impact on the density of subgraphs.

Another reasonable evaluation strategy was to observe the density of subgraphs. Bipartite graph had a special structure differs from ordinary graphic structures. Therefore, it was necessary to observe the extracted dense subgraphs separately. Table III concluded the features of each maximum *quasi*-truss.

TABLE III. FEATURES OF MAXIMUM QUASI-TRUSS

File name	node	edge	size(kb)	#T
cmuDiff	240	477	2.95	238
cmuSame	392	781	5.27	390
cmuSim	114	225	1.54	112
HetRac	353	703	3.88	351
MovieLens	225	447	2.58	223

The node and edge indicated the number of nodes and the number of edges in each dense subgraph respectively. The size was the total amount of subgraphs defined as $(|V_1| \cup |V_2|, |e| \cup |e'|)$. The $\#T$ indicated the number of triangles anchored in each dense subgraphs. In the case of bipartite graph, a subgraph had the density one if and only if it was a biclique according to the concluded definition in [16]. The definition of density for bipartite graph in [16] cannot be adopted directly to estimate the density of *quasi*-trusses in a bipartite graph for it was based on triangular structure. Thus, we simply addressed the amount ratio that compaired the size of dense subgraphs with their matrix graphs containing all triangles. Moreover, we also compaired the number of triangles of the dense subgraps with their matrix graphs containing all triangles in order to observe the ratio of number of triangles. Then, analyzed the relationship between the amount ratio and the ratio of the number of triangles based on the statistic results shown as Fig. 4.

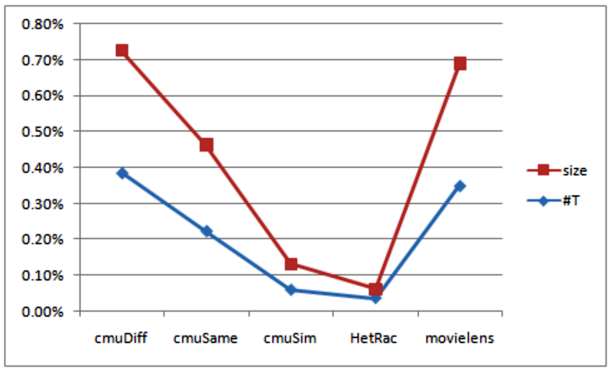

Fig. 4. Relationhsip between size and the number of triangles

Each point in the red graph illustrated the percentage of the amount of the largest subgraphs in each bipartite graph. Each point in the blue graph illustrated the percentage of the number of triangles of the largest subgraphs in each bipartite graph. From Fig. 4, the amount of the dense subgraphs increased linearly with the number of triangles anchored in the dense subgraphs. These statistical results also proved the previous experimental results in Table II.

VI. CONCLUSION

We implemented the algorithm for *quasi*-truss, which was a novel notion of dense subgraph in a bipartite graph introduced in [3]. This notion was an expanded version of *k*-truss decomposition [2]. An effective algorithm was also introduced for *quasi*-truss decomposition in a bipartite graph. We verified the scalability of our algorithm by experiments on real-world datasets. The results showed a significant effectiveness on decomposing a bipartite graph based on triangle structure.

We plan to research the theoritical proof for the density evaluation of bipartite graph by adopting the *quasi*-truss decomposition algorithm, and time complexity for dense subgraph extraction as the future perspective. Furthermore, as one of triangle's properties, the research of clustering coefficient of a bipartite graph is available to analyze the connectivity situation.

REFERENCES

[1] J. Cohen, Truss: cohesive subgraphs for social network analysis, 2008.

[2] J. Wang and J. Cheng, Truss decomposition in massive networks, VLDB2012, 2012, pp. 812-823.

[3] Y. T. Li, Kuboyama, and H. Sakamoto, Mining twitter data: discover quasi-truss from bipartite graph, 5th International Conference on Intelligent Decision Technologies, to appear

[4] G. W. Flake, S. Lawrence, and C. L. Giles, Efficient identification of web communities, KDD2000, 2000, pp. 150-160.

[5] J. M. Kleinberg, Authoritative sources in a hyperlinked environment, SODA1998, 1998, pp. 668-677.

[6] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins, Extracting large-scale knowledge bases from the web, VLDB1999, 1999, pp. 639-650.

[7] J. Abello, M. G. C Resende, and S. Sudarsky, Massive quasi-clique detection, LATIN2002, 2002, pp. 598-612.

[8] H. Matsuda, T. Ishihara, and A. Hashimoto, Classifying molecular sequences using linkage graph with their pairwise similarities, Theor. Compt. Sci., 1999, 210(2)305-325.

[9] J. Pei, D. Jiang, and A. Zhang, On mining cross-graph quasi-cliques, 2005, SIGKDD.

[10] J. Cohen, Graph twiddling in a mapreduce world, Computing in Science and Engineering, 2009, 11(4)29-41.

[11] S. B. Seidman, Network structure and minimum degree, Social Networks, 1983, 5(3)269-287.

[12] 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

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

[14] 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.

[15] J. Cheng, Y. Ke, A. W. C. Fu, J. X. Yu, and L. Zhu, Finding maximal cliques in massive networks, ACM Transactions on Database Systems, 2011, 36(4) Article No. 21.

[16] J. Chen and Y. Saad, Dense subgraph extraction with application to community detection, Knowledge and Data Engineering, IEEE Transaction, 2012, Volume:24, Issue:7.

[17] H. Y. Zha, X. F. He, C. Ding, M. Gu, and H. Simon, Bipartite graph partitioning and data clustering, 2001, Proceedings of ACM CIKM 2001.

[18] L. Danon, A. Diaz-Guilera, J. Duch, and A. Arenas, Comparing community structure identification, Journal of Statistical Mechanics: Theory and Experiment, 2005, Vol. 2005, p. P09008.

 ISBN: 978-1-61208-311-7