K-Truss Based Top-Communities Search in Large Graphs

Anzuruni Maulidi Katunka, Cairong Yan(✉), Kossonon Brandou Serge, Zhaohui Zhang
School of Computer Science and Technology
Donghua University
Shanghai, China
akatunka@yahoo.com, cryan@dhu.edu.cn, kossononserge@yahoo.fr, zhzhang@dhu.edu.cn

***Abstract*— The study of community search problem subjected to find densely connected subgraphs that satisfy the query conditions in social and information networks has attracted many researchers. In most real life networks a query node normally involves in several communities, therefore, most likely the same query results into communities of different densities. In this paper, based on the concept of *k*-truss, which is a type of cohesive subgraph, we propose linear-time algorithms to find the top-*r* *k*-truss communities containing a query node in large graphs. We adapt some optimal index construction strategies which can support efficient search of top-r k-truss communities with a linear cost to work for top-r community search. We carry out extensive experiments on real-world large networks, and the results signify the efficiency and effectiveness of the proposed algorithms.**

Keywords*— *k-truss, community search, large graph, algorithm

I. INTRODUCTION

Graphs are one of the most powerful data structures useful in a range of applications. In recent years there has been a vast interest in storing and managing graphical data. In many real-world networks such as communication networks, social media, biological and chemical networks, graphical representation of data is used. For example, chemical data is modeled as a graph by assigning atoms as nodes and bonds as edges between them. Biological data is represented the same way, only with amino acids as the nodes and links between them as edges. Also in biological ecosystem, a food web is depicted graphically showing feeding connections among species of an ecological community.

In graphical representation of data, nodes connect each other and form communities. Nodes in one community tend to connect each other than the rest of nodes in network. In a social network, an entity can relate with other entities and form cohesive structures which in turn may form different communities. For example, the person can form a community with university classmates which might be considerably different from the community formed by his or her family members which can also be totally different from the one formed by his or her workmates. Thus, community structure can provide valuable information of the network.

Different community models have been studied based on different dense subgraph structures such k-truss [1],[2],[3]; k-core [4],[5],[6] ; quasi-clique [7],[8],[9] to mention the few. In this paper, we use k-truss as the main measurement tool which is a type of cohesive subgraph defined based on triangle which models the stable relationship among three nodes. Given a graph G, k-truss community of G is the largest subgraph in which every edge is contained in at least $(k - 2)$ triangles within the subgraph. However, using k-truss for the same value of k a vertex is contained in at most one k-truss community hence opposing the reality that a vertex can participates in more than one community. Huang et al [1] proposed a community model based on the notion of k-truss by imposing edge connectivity constraint on it, that is, any two edges in a community either belong to the same triangle or are reachable from each other through a series of adjacent triangles. Edge connectivity constraint ensures the community is not only cohesive and connected but also enables possibility of a vertex to participate in multiple communities.

A clique of order k is a k-truss [10]. The number of degree for each vertex in k-clique is k-1 while for k-truss is at least k-1. Therefore k-clique is the relaxing model of k-truss. The highest possible number of communities(max(n(C)) a node can participate can be calculated by taking degree of query node (deg(v)) divide by the minimum number of degrees (k-1) that can be used to form a community expressed as:

$$Max\ (n(C)) = deg\ (v) / (k\text{-}1) \quad \textbf{(1)}$$

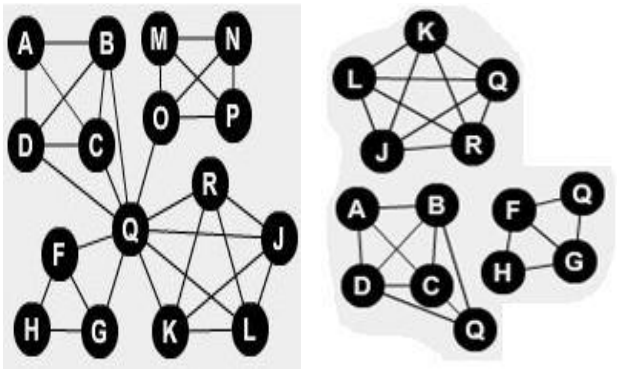
For example, if a query node has 15000 degree, the possible number of communities is at most 7500 and 3750 for k=3 and 5 respectively using (1). Therefore in our study, we propose the methods of finding the top-r k-truss communities rather than finding all communities which has got some useful applications. In Fig. 1 query node Q contains three 3-truss communities C_1 = {K, J, L, Q, R}, C_2= {A, B, C, D, Q} and C_3 = {F, G, H, Q}. Among these three communities, C_1 may qualifies also for 4-truss and 5-truss while C_2 may qualify also for 4-truss but C_3 qualifies only for 3-truss.The higher the value of k the higher strong relationships among community members and hence the higher the preferences in finding top-r k-truss communities.

In this paper, we put forward k-truss with edge connectivity (KTEC) model for top-r communities(i.e top-r KTEC). The main contributions of this paper are as follows:

- We introduce top-r k-truss community search problem. This problem arises when k-truss with edge connectivity constraint is used.

✉ Corresponding author: Cairong Yan (cryan@dhu.edu.cn)

978-1-5386-1072-5/17 $31.00 © 2017 IEEE
DOI 10.1109/CBD.2017.49



(a) Toy social Graph G (b) Three 3-truss communities contain Q

Fig. 1. K-truss community example

- We propose efficient algorithms for searching top-r k-truss communities containing a query node. We also show how previous proposed index constructions methods can be used with these algorithms.
- We perform comprehensive experiments to test the performance of our proposed algorithms on several data sets of different sizes to demonstrate the efficiency and effectiveness of our algorithms.

The rest of this paper is organized as follows: problem definition is given in Section II. In Section III, we introduce our proposed algorithms and implementation framework. Then, experimental results and analysis are presented in Section IV. Related works are introduced in Section V and we conclude the paper in Section VI.

II. PROBLEM DEFINITION

In this section, we define the problem of finding top-r k-truss communities more formally as well defining some important terms related to K-truss community search.

In this study undirected, unweighted simple graph $G = (V, E)$ is considered with $n = |V|$ vertices and $m = |E|$ edges. Table 1 gives the summary of some important notations and descriptions. The following are definitions of some important terms as defined in wang et al [1].

TABLE 1. Frequently used Notations

Notation	Description
G (V, E)	Undirected, unweighted, simple graph
n	Number of vertices in G i.e |V|
m	Number of edges in G i.e |E|
N(v)	The set of neighbors of vertex v in G
deg(v)	Degree of vertex v in G. i.e |N (v)|
Sup(e)	Support of an edge e (u, v) in G
sup (e, H)	Support of an edge e in subgraph H.
τ (e)	Trussness of an edge e
τ (H)	Trussness of subgraph H

Definition 1 (Support). Support of an edge e is the number triangles share it.

Definition 2 (Triangle Adjacency). Two triangles are adjacent if they share common edge.

Definition 3 (Triangle Connectivity). Two triangles are connected if they are adjacent or can be reached by a series of adjacent triangles.

Definition 4 (Edge Connectivity). Two edges obey edge connectivity if they are in the same triangles or their triangles are connected.

Definition 5 (Subgraph Trussness). The trussness of a subgraph $H \subseteq G$ is the minimum support of an edge in H, denoted by $\tau(H) = \min \{\sup(e, H): e \in E(H)\}$.

Definition 6 (Edge Trussness). The trussness of an edge $e \in E(G)$ is defined as $\tau(e) = \max_{H \subseteq G} \{\tau(H): e \in E(H)\}$.i.e the trussness of the maximum subgraph trussness it is contained. Fig. 2 shows edges and their corresponding trussness.

Definition 7 (K-Truss Community). Given a graph G and an integer $k \geq 2$, G' is a k-truss community if it satisfies the following conditions: 1. K-truss: G' is a subgraph of G, denoted as $G' \subseteq G$, such that $\forall e \in E(G')$, $\sup(e, G') \geq (k-2)$ 2: Edge connectivity: for $\forall e_1, e_2 \in E(G')$, they are triangle connected. 3. Maximal subgraph: G' is the maximal subgraph if satisfies conditions (1) and (2) and there is no subgraph in G which is the superset of it.

Problem 1: Given a graph G (V, E), a query vertex $v_q \in V$ and an integer $k \geq 2$, the problem is to find top-r k-truss communities containing v_q. The top most communities are the ones which contain the higher subgraph trussness 'k'.

Example 1: For the graph shown in Figure 1 (a), suppose we want to find top-3 3-truss communities which contain node Q, where r = 3 and k= 3. We have three 3-truss communities which satisfy query conditions: C_1 ={J, K, L, Q, R}, C_2 = {A, B, C, D, Q} and C_3 = {F, G, H, Q}. The minimum edge supports (subgraph trussness) of C_1, C2 and C3 are 5, 4 and 3 respectively. Therefore C1, C2 and C3 are top-1, top-2 and top-3 respectively.

Example 2: For the graph shown in Figure 1 (a), suppose we want to find top-3 4-truss communities which contain node *Q, where* r = 3 and k=4. We have only two 4-truss communities which satisfy query conditions: C_1={J, K, L, Q, R}, C_2 = {A, B, C, D, Q}.

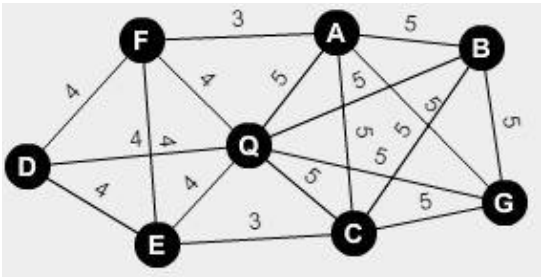


Fig. 2. The graph showing edges' trussness

III. FRAMEWORK AND ALGORITHMS

To enable querying of top-r k-truss communities on a graph we build *k*-truss index after which the proposed Query processing algorithms are designed.

A. Index construction

Wang et al [13] proposed trussness decomposition algorithm in massive networks. Later on, Huang et al [1] use this algorithm to construct index used for k-truss communities processing. We also apply this method to construct simple k-truss index to support query processing algorithms. The outline of the truss decomposition algorithm process is given without writing the algorithm in this paper. The complete trussness decomposition algorithm is in [1] and [13]. Initially, the algorithm computes the trussness of each edge found in graph G**.** Then hash table is used to keep all the edges and their trussness. For each vertex $v \in V$, its neighbors $N(v)$ are sorted in descending order of the edge trussness $\tau(e(u, v))$ for $u \in N(v)$. For each distinct trussness value $k \geq 2$, the position of the first vertex u is marked *in* the sorted adjacency list where $\tau(e(u, v)) = k$. This supports efficient retrieval of incident edges of v, starting with a certain highest trussness value. This is named as simple k-truss index.

Huang et al [1] revealed two drawbacks of simple k-truss index when is used for query processing which lead to unnecessary repetition of accessing qualified edges and disqualified edges and propose a novel Triangle Connectivity Preserved Index or TCP-Index in short, which avoids those computational problems. This strategy is adapted in this study. First, the algorithms perform truss decomposition for G (V, E). For each vertex $x \in V$, graph G_x is built, where $V(Gx) = N(x)$, and $E(Gx) = \{(y, z) | (y, z) \in E(G), y, z \in N(x)\}$. Then the weight $w(y, z) = \min\{T((x, y)), \tau((x, z)), \tau((y, z))\}$ is assigned for each edge $(y, z) \in E(Gx)$, because triangle xyz can appear only in k-truss community where $k \leq w(y, z)$. The TCP-Index for vertex 'x' is a tree structure, denoted as T_x, which is initialized to be the node set N(x). Then for each k from the largest weight k_{max} to 2, an iteration to collect the set of edges $S_k \subseteq E(Gk)$ whose weight is k is performed. For each $(y, z) \in S_k$, if y, z are still in different components of T_x, the edge (y, z) with a weight w(y, z) is added into T_x. Basically, T_x is the maximum spanning forest of G_x. The collection of trees T_x for all $x \in V$ form the TCP-Index of graph G.The complete algorithm and more illustration in [1].

B. Query processing

Huang et al [1] proposed the query processing algorithm using simple k-truss index which takes $O(d_{Amax}|Ans|)$ time to process one query, where Ans is the union of the produced k-truss communities, |Ans| is the edge number in Ans and d_{Amax} is the maximum vertex degree in Ans. The drawback of this algorithm is that every edge in a k-truss community is accessed atleast 2(k-2) times. Due to this drawback, the more efficient query processing algorithm which uses TCP-Index was proposed by which every edge is accessed exactly twice. The time complexity of this Algorithm is O(|Ans|), where Ans is the union of the produced k-truss communities and |Ans| is the number of edges in Ans. However, both algorithms find all communities contain query nodes with equal priority. We adapt both techniques and improve them so as to support top-r k-truss communities searching and propose algorithms 1 and 2.

Algorithm 1 Top-r query processing using simple k-truss index

Input: G = (V, E), integers k and r, query vertex v_q
Output: top-r k-truss communities containing v_q

```
1: visited ← Ø; s ← 0;
2: for u ∈ N (v_q) do
3:   if max (τ ((v_q, u)) ≥ k and (v_q, u) ∉ visited
4:     s ← s + 1;
5:     if s ≤ r then
6:       C_s ← Ø; Q ← Ø;
7:       Q.push ((v_q, u)); visited ← visited ∪{(v_q, u)};
8:       while Q ≠ Ø
9:         (x, y) ← Q.pop (); C_s ← C_s ∪{(x, y)};
10:        for z ∈ N(x) ∩ N(y) do
11:          if τ ((x, z)) ≥ k and τ ((y, z))
12:              assume, τ (x, z) ≥ τ(y, z)
13:            if (x, z) ∉ visited
14:              Q.push ((x, y)); visited ← visited ∪ {(x, z)};
15:            if (y, z) ∉ visited
16:              Q.push ((y, z)); visited ←visited ∪ {(y, z)};
17: return {C_1 . . . C_r};
```

1) Query processing using simple k-truss index

The straight forward method is to find all communities and then find top-r communities but this is very costly and hinders efficiency. So we propose the algorithms which can find top-1, top-2, top-3 and so forth. In algorithm 1, given integers k and r and query vertex v_q, an edge (v_q, u) with highest trussness is pushed into a queue Q and performs a BFS traversal to search for other edges for expanding C_s. Each iteration may give a seed which will form new community. To find top-r K-truss community BFS is performed starting with the seed with highest trussness. This is controlled in line 3. When Q becomes empty, all edges in C_s have been found. Then the algorithm checks the next unvisited incident edge of v_q with highest k and uses it as a new seed to form new community C_{s+1}. This process iterates until all incident edges of v_q have been processed. Finally, a set of top-r k-truss communities containing v_q are returned. The time complexity of algorithm 1 is $O(d_{Amax}|Ans_r|)$ where Ans_r is the union of the produced top-r k-truss communities, $|Ans_r|$ is the edge number in Ans_r and d_{Amax} is the maximum vertex degree in Ans_r.Every edge in top-r k-truss communities is still accessed atleast 2(k-2) times.

Example 3: For the graph shown in Fig. 2, suppose we want to query top-3 4-truss communities which contain node Q, where r = 3 and k=4. Algorithm 1 visits the edge with highest trussness formed by neighbors of vertex Q. From the graph $\text{Max}(\tau(e)) = 5 \geq 4$ so any edge whose $\tau(e) = 5$ qualify to be a seed edge. In this case let edge (Q, B) to be the first seed. The algorithm adds it into the queue. The algorithm pops up (Q, B) and inserts it into community C_s. Then the algorithms check the common neighbors of 'Q' and 'B' and the edges between them. Consider the common neighbor 'D' as example. As τ

((Q, D)) = 5 ≥ 4 and τ ((B, D)) = 5 ≥ 4 both edges are added into the community C_s and then pushed into the queue for further expansion. This BFS expansion process continues until the queue is empty then C_s is returned. C_1 = {Q, A, B, C, D}.since r = 3, the algorithm checks if there still exists unvisited edges of Q with τ (e) >= 4 and visits the edge with highest trussness. Let's take e (Q, F) as second seed because τ ((Q, F)) = 4 ≥ 4. The algorithm adds it into the queue and the process repeats until all edges to form C_2 are obtained and C_2 is returned. C_2 = {Q, F, G, E}.Thus C_1 = {Q, A, B, C, D}, C_2 = {Q, F, G, E} and C_3 = Ø because no more unvisited edge e with τ (e) ≥ 4.

2) Query processing using TCP-index

Similar to Algorithm 1, Algorithm 2 computes the k-truss communities for a query vertex v_q by expanding from each incident edge on v_q in a BFS manner. To find top-r k-truss community BFS is performed starting with the seed with highest trussness. This is controlled in line 3. If there exists an unvisited edge (v_q, u) with τ ((v_q, u)) ≥ k, (v_q, u) is the seed edge to form a new community C_s (line 2-4). Then the algorithm performs a BFS traversal using a queue Q in line 5-14. For an unvisited edge (x, y), it searches the vertex set V_k(x, y) to from T_x. For each z ∈ V_k(x, y), the edge (x, z) is added into C_s. Then it performs the reverse operation, i.e., if (z, x) is not visited yet, it is pushed into Q for z-centered community expansion using T_z. Note that (z, x) and (x, z) are considered different in a queue and in the set of visited but they are considered the same in the community.The process iterates until all incident edges of v_q are processed. When Q becomes empty, all edges in C_s have been found. Finally, a set of top-r k-truss communities containing v_q are returned. The time complexity of this Algorithm is O(|Ans_r|), where Ans_r is the union of the produced top-r k-truss communities and |Ans_r| is the number of edges in Ans. Using this method every edge is still accessed exactly twice in top-r communities.

Algorithm 2 Top-r query processing using TCP-Index

Input: G = (V, E), integers k and r, query vertex v_q
Output: top-r k-truss communities containing v_q

```
1: visited ← Ø; s ← 0;
2: for u ∈ N (v_q) do
3:   if max (τ ((v_q, u)) ≥ k and (v_q, u) ∉ visited
4:     s ← s + 1;
5:     if s ≤ r then
6:         C_s ← Ø; Q ← Ø;
7:         Q.push ((v_q, u));
8:       while Q ≠ Ø
9:           (x, y) ← Q.pop ();
10:          if (x, y) ∉ visited
11:             compute V_k (x, y);
12:             for z ∈ V_k (x, y) do
13:               visited ← visited ∪ {(x, z)}; C_s ← C_s ∪ {(x, y)};
14:                   if the reverse edge (z, x) ∉ visited
15:                     Q.push ((z, x));
16: return {C_1 … C_r};
```

[2] http://snap.stanford.edu/data/email-Enron.html
[3] http://snap.stanford.edu/data/com-Amazon.html
[4] http://snap.stanford.edu/data/com-DBLP.html

IV. EXPERIMENTS

In this section, we present experimental study and evaluate the effectiveness and efficiency of top-r k-truss communities search using two methods. The first method uses algorithm 1 which is supported by simple k-truss index and the second method uses algorithm 2 which is supported by TCP-index.

A. Experimental setup

We run all the experiments on a PC with Intel 2.4GHz 4-Cores and 16GB memory running 64-bit Windows 8. All algorithms were implemented in java.

Four real-world networks are used as datasets in experiments, and the statistics are shown in Table 2. Facebook dataset is an anonymized dataset which consists of friends lists from Facebook[1]. Nodes represent user and edges between them represent user relationship. Email-Enron dataset is the Enron email communication network that covers the email communication[2]. Nodes of the network are email addresses and if an address *i* sent at least one email to address *j*, the graph contains an undirected edge from *i* to *j*. Amazon is a product co-purchasing network of Amazon website[3]. Nodes in Amazon represent products and if two products are frequently co-purchased, there exists an edge between them. DBLP is a scientific coauthor network extracted from a recent snapshot of the DBLP database[4]. Nodes in DBLP graph represent authors, and edges represent collaboration relationship.

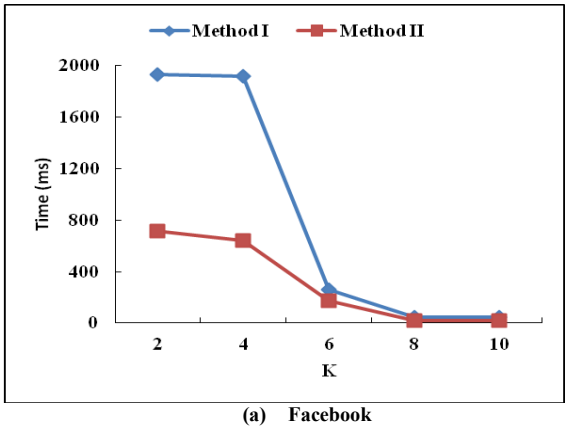
B. Performance

We evaluate and compare the performance of the two top-r *k*-truss community algorithms: Algorithm 1 which uses the simple *k*-truss index and Algorithm 2 which uses the TCP-Index, with different values of parameters k and r. We refer the use of algorithm 1 and algorithm 2 as method I and method II respectively in all Fig. 3, 4 and 5 which show results.

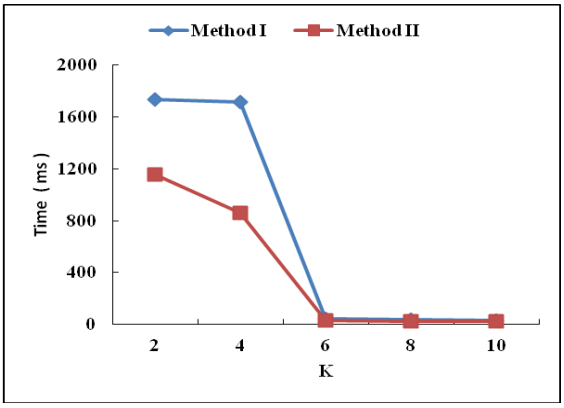
In the first experiment, we select query vertex with the highest degree from Facebook and Email-Enron datasets to test effect of k on query processing time in top-r k-truss communities search. To run the queries, fixed value of r = 3 was used while varying k = 2, 4, 6, 8 and 10.To minimize the effect of system load on query time we repeat each query ten times before taking the average time. For both datasets, we run the query for both Method I and Method II. The average query processing time for each dataset is presented by graphs in Figure 3. The results show that query processing time decreases as the value of *k* increases because the number of resulted communities become smaller when *k* increases. When k = 2 all edges connected with query vertices form single community and when the whole graph is connected the result of the query becomes the whole graph G.

TABLE 2. Datasets

Dataset	# Nodes	# edges	# triangles
Facebook	4039	88,234	1,612,010
Email-enron	36692	183,831	727,044
Amazon	334,863	925,872	667,129
DBLP	317,080	1,049,866	2,224,385

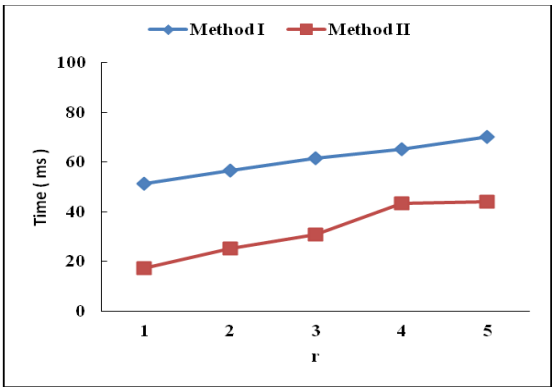


(a) Facebook

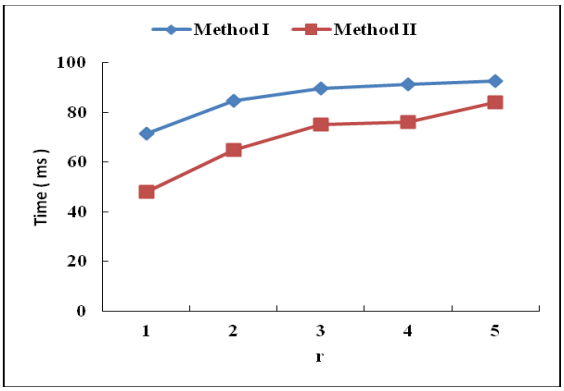


(b) Email-Enron

Fig. 3: Query time for different values of k in different datasets

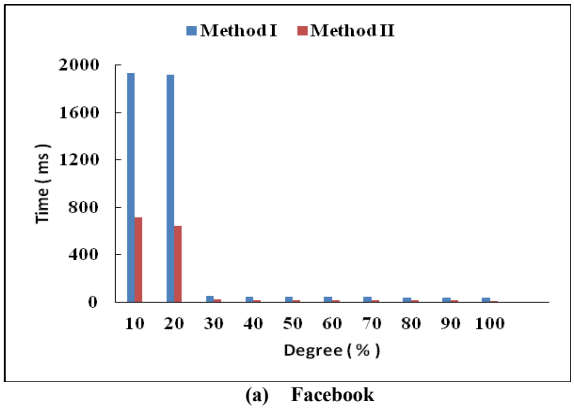


(a) Facebook

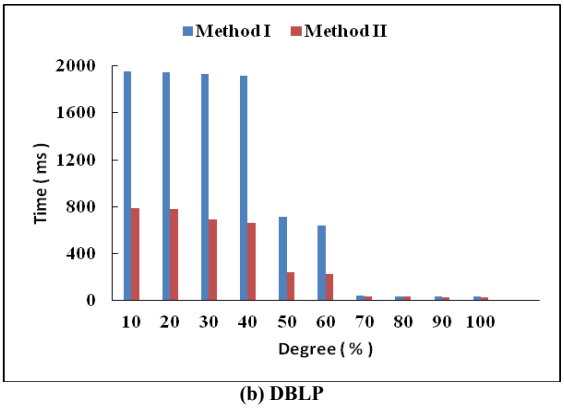


(b) Amazon

Fig. 4: Query time for different values of r in different datasets



(a) Facebook



(b) DBLP

Fig. 5: Query time of two methods for data sets of different size and vertices' degree

In the second experiment, we also select query vertex with the highest degree from each dataset to test the effect of parameter r on query processing time. For each network, we run the query with the fixed value of k = 5 while varying r = 1, 2, 3, 4 and 5.We repeat each query ten times then we take the average time. To test the effect of parameter r we run the query using method I and method II in Facebook and Amazon. The average query processing time for each dataset is presented by graphs in Fig. 4. We can easily see that query time increases when r increases using both method I and II in both datasets.

In the third experiment, we test query time on Facebook and DBLP networks using query vertices with different degrees. For both networks, we sort the vertices in descending order of their degrees and partition them into 10 buckets before selecting 10 query vertices randomly from each bucket. In all cases we fix $k = 10$ and r = 3. The results in Fig. 5 show that the average query time is larger for the high degree query vertices because they normally have large-denser *k*-truss communities. Since larger graphs might be denser than small graphs, in query processing the effect of graph size to the query time is trivial. For example, Facebook dataset has the least number of vertices and edges but has larger number of triangles than Email-Enron and Amazon.

V. RELATED WORK AND DISCUSSION

This work is related to global community detection which intends to find all communities in the graph, and local community search which involves searching of community(s) containing given query node(s) in network.

Different works related to community detection have been studied. Palla et al [11] addressed overlapping community structure that exists in many real networks. They proposed a clique percolation method (CPM), in which a community was defined as a *k*-clique component. Cui et al. [7] studied the problem of online search of overlapping communities for a query vertex by designing a relaxing model of k-clique named as α-adjacency *γ*-quasi-*k*-clique model later on Shan et al [8] studied the problem of searching overlapping communities for group query based also on α-adjacency *γ*-quasi-*k*-clique. Li et al [4] worked on Influential Community Search in Large Network. Given a graph G = (V, E), parameters k and r and the weight vector W, the problem was to find top-*r* *k*-influential communities in a network where k is the minimum degree that determines the cohesiveness of the graph adopted from k-core. This work is different from ours as it deals with global community detection where no any query node is required. Also, it adopted k-core mode which based on minimum degree. The other different is that the influence of the community depends also on weight vector which is contrary to our work where the popularity of the nodes depends on the number of triangles it forms which signifies strong relationship in the graph.

Compared to global community detection, local community search is more light-weight and flexible as it only needs to explore a part of the graph around the query nodes rather than the whole graph, the property which makes it more suitable for online querying. Due to its advantages local community search, has also received a lot of attention [1], [2], [5], [6] and [9]. Our work is more inspired by Huang et al. [1] who proposed a community model based on the k-truss concept to query all k-truss communities for single query node in large graphs.

Method I and II use the proposed algorithms 1 and 2 which have the time complexity of O ($d_{Amax}|Ans_r|$) and O ($|Ans_r|$) respectively. The results, signifies that method I is more efficient than method II. Also, the tendency of processing time to increase as r increases in experiment 2 signifies that our methods are more efficient than existing methods which find all communities. In many real application domains, finding the most ranking communities is highly desired. For example, in co-authorship network of the database community, to be aware of the recent trends of database research, the administrator may want to know the most strongly related research groups (communities) a certain author participates. Different scenarios may also apply in social networks domain.

VI. CONCLUSION

In this paper, we study the top-r k-truss community search in large graphs. We propose the k-truss with edge connectivity (KTEC) model for top-r communities for single query node. The proposed algorithms for top-r k-truss communities search can work for both static and dynamic graphs as long as the constructed index can support edge insertion and deletion in the graph. We conduct extensive experiments on large real-world networks, and the results demonstrate the effectiveness and efficiency of the proposed algorithms. We plan to improve our algorithms to work in distributed environment as well as to deal with top-r k-truss communities search for group queries.

ACKNOWLEDGMENT

This work was supported by the National Natural Science Foundation of China No. 61402100 and 61472004, the Fundamental Research Funds for the Central Universities of China No. 17D111205, and the Online Education Fund of the Ministry of Education No. 2017YB112.

REFERENCES

[1] X. Huang, H. Cheng, L. Qin, W. Tian and J.X. Yu, "Querying k-truss community in large and dynamic graphs," ACM SIGMOD international conference on Management of data. 2014.

[2] X. Huang, L.V.S. Lakshmanan, J.X. Yu, and H. Cheng, "Approximate closest community search in networks," VLDB Endowment, 2015. 9(4): p. 276-287.

[3] J. Cohen, "Trusses: cohesive subgraphs for social network analysis", National Security Agency Technical Report, 2008: p. 16

[4] R.H. Li, L. Qin, J.X. Yu and R. Mao, "Influential community searches in large networks", VLDB Endowment, 2015. 8(5): p. 509-520.

[5] M. Sozia, and A. Gionis, "The community-search problem and how to plan a successful cocktail party," The 16th ACM SIGKDD international conference on Knowledge discovery and data mining. 2010. ACM.

[6] W. Cui, Y. Xiao, H. Wang, and W. Wang, "Local search of communities in large graphs," Proceedings of the 2014 ACM SIGMOD international conference on Management of data. 2014. ACM.

[7] W. Cui, Y. Xiao, H. Wang, Y. Lu and W. Wang, "Online search of overlapping communities," Proceedings of the 2013 ACM SIGMOD international conference on Management of data. 2013. ACM.

[8] J. Shan, D. Shen, T. Nie, Y. Kou and G. Yue, "An efficient approach of overlapping communities search," International Conference on Database Systems for Advanced Applications. 2015. Springer.

[9] J. Shan, D. Shen, T. Nie, Y. Kou and G. Yue, "Searching overlapping communities for group query," World Wide Web, 2015: p. 1-24.

[10] G. Palla, I. Der´enyi, I. Farkas, T. Vicsek, "Uncovering the overlapping community structure of complex networks in nature and society," Nature 435(7043), 814–818 (2005).

[11] B. Adamcsek, G. Palla, I.J. Farkas, I. Der´enyi and T. Vicsek, "Cfinder: locating cliques and overlapping modules in biological networks," Bioinformatics 22(8), 1021–1023 (2006).

[12] J.P. Bagrow and E.M. Bollt, "Local method for detecting communities," Phys. Rev. E 72(4), 046108 (2005).

[13] J. Wang and J. Cheng, "Truss decomposition in massive networks" Proceedings of the VLDB Endowment, 2012. **5**(9): p. 812-823