Data Mining and Knowledge Discovery manuscript No.
(will be inserted by the editor)

Fast Truss Community Query in Large-scale Dynamic Graphs

Zheng Lu · Yunhe Feng · Qing Cao

Received: date / Accepted: date

Abstract Recently, there has been significant interest in the study of the community search problem in social and information networks: given one or more query nodes, find densely connected communities containing the query nodes. However, most existing algorithms require linear computational time to the size of the found community for each specific K value. Therefore, state-of-the-art algorithms have limited scalability in large scale graphs, where communities grow to millions of edges.

In this paper, given an undirected graph G and a set of query nodes Q, we study community query using the k-truss based community model. We formulate our problem of finding a connected truss community, as finding a connected k-truss subgraph with all possible k that contains Q. The state-of-art approximation algorithm can achieve this goal with a time complexity of $O(n'm')$ where n' and m' are the size of the result truss community. For queries that only identity and exact size of communities are required, We construct an index structure that can retrieve there information of all connected k-truss communities that contain Q with all possible K values. The algorithm can run in $\sum_{u \in Q} d(u)$, where $d(u)$ is the degree of vertex u. We prove that this is the optimal time complexity for truss community query. Extensive experiments on real-world networks show the effectiveness and efficiency of our algorithms.

Keywords K-truss · dynamic graph · query process

Zheng Lu
Electrical Engineering & Computer Science, University of Tennessee
E-mail: zlu12@vols.utk.edu

Zheng Lu
Electrical Engineering & Computer Science, University of Tennessee
E-mail: yfeng14@vols.utk.edu Qing Cao
Electrical Engineering & Computer Science, University of Tennessee
E-mail: cao@utk.edu

1 Introduction

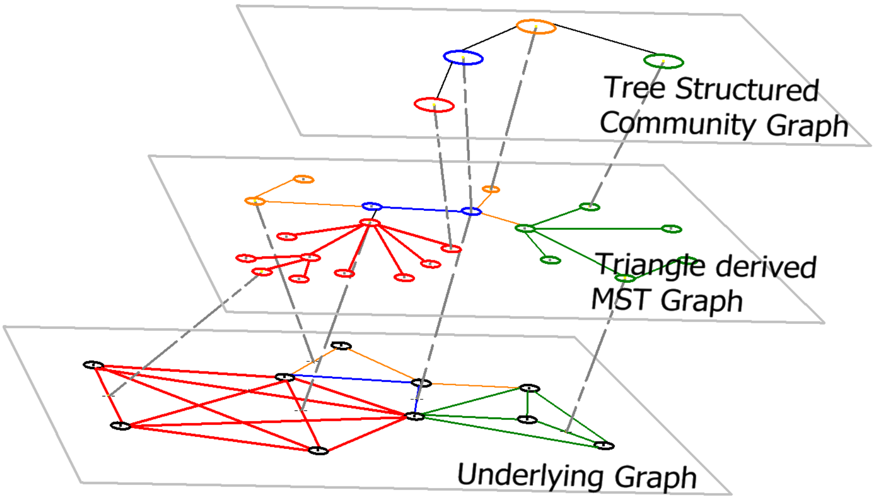


Fig. 1 Two layer index structure for k-truss community queries.

Community structures naturally exist in many real-world networks such as social, biological, collaboration, and communication networks. The task of community detection is to identify all communities in a network, which is a fundamental and well-studied problem in the literature. Recently, several papers have studied a related but different problem called community search, which is to find the community containing a given set of query nodes. The need for community search naturally arises in many real application scenarios, where one is motivated by the discovery of the communities in which given query nodes participate. Since the communities defined by different nodes in a network may be quite different, community search with query nodes opens up the prospects of user-centered and personalized search, with the potential of the answers being more meaningful to a user Huang et al (2014). As just one example, in a social network, the community formed by a person's high school classmates can be significantly different from the community formed by her family members which in turn can be quite different from the one formed by her colleagues McAuley and Leskovec (2012).

Various community models have been proposed based on different dense subgraph structures such as k-core Sozio and Gionis (2010); Cui et al (2014); Li et al (2015), k-truss Huang et al (2014), quasi-clique Cui et al (2013), weighted densest subgraph Wu et al (2015), to name a few major examples. Of these, the k-truss as a definition of cohesive subgraph of a graph G, requires that each edge be contained in at least (k - 2) triangles within this subgraph. It is well known that most of real-world social networks are triangle-based, which always have high local clustering coefficient. Triangles are known as the fundamental building blocks of networks Wang and Cheng (2012). In a social network, a triangle indicates two friends have a common friend, which shows a strong and stable relationship among three friends. Intuitively, the more common friends two people have, the stronger their relationship. In a k-truss, each pair of friends is "'endorsed"' by at least (k - 2) common friends. Thus, a k-truss with a large value of k signifies strong inner-connections between members of the subgraph. Huang et al. Huang et al (2014) proposed a community model based on the notion of k-truss as follows. Given one query node q and a parameter k, a k-truss community containing q is a maximal k-truss containing q, in which each edge is "'triangle connected"' with other edges. Triangle connectivity is strictly stronger than connectivity.

The proposed TCP-index Huang et al (2014) works well to find overlapping communities containing a single query vertex q. However, it is not obvious how to support more complex queries involving multiple query vertices with the index. For example, queries such as "Does a query set of n vertices $q_1, ..., q_n$ belong to the same k-truss community?" require n individual k-truss community search and then find intersections among the resulting communities. Although the query process for TCP-index is theoretically optimal for a single vertex, which has a time complexity of $O(n'm')$ where n' and m' is the size of the result truss community, it faces scalability issues in large-scale real-world graphs where communities can grow to millions of edges.

In this paper, we aim to develop a novel 2-level index structure to efficiently answer complex multiple-vertex queries that can easily be used as building blocks for many applications. The query process takes a different, more natural approach that it first identify the resulting communities with the top-level index that do not require community search on any query vertices. Then, if the query require the exact community rather than community relations among query vertices, the query process discovers the exact communities by a single run of breath first search on the bottom-level index, which is proved to be theoretically optimal and it is efficient in practice. With this approach, the index can easily answer various kinds of queries such as "What is the k-truss community with highest k that a set of query vertices belong to" (Max-k truss query) and "Show all the k-truss communities that a set of query vertices belong to" (Any-k truss query) as well as multiple-vertex k-truss query.

Finding out the identity of the queried k-truss community and then doing the exact community search is also more time and space efficient. As a query vertex can belong to multiple k-truss communities, our top-level index is build on edges of the graph as each edge can only belong to one k-truss community with a same k. The query algorithm for k-truss community identity search can run in $\sum_{u \in Q} d_u$ with the top-level index, where d_u is the degree of a query vertex u. To retrieve the exact community, our query algorithm can find it in linear time complexity to the community size with the bottom-level index, which is the optimal time complexity.

We construct our index structure in a bottom-up manner. The bottom-level index can be efficiently constructed from the underlying graph by representing edges with vertices and representing triangle connectivity with edge connectivity in the transformed graph, and find the minimum spanning tree of the transformed graph to form the bottom-level index. Although the transformed graph may have number of edges equal to three times the number of triangles in the graph, our index construction algorithm is efficient in the sense that it can generate the minimum spanning tree during the tranform without store the transformed graph, so the time and space complexity is only $O(m \log m)$ and $O(m)$ respectively where m is the number of edges in original graph. After the bottom-level index has been constructed, the top-level index can easily being constructed on top of it with a single run of breath first search, where both the space and time complexity is $O(m)$.

Our contribution can be summarized as follows,

- We introduce the k-truss community identity search, which is supported by various query types. The k-truss community identity search is efficient for many real-world applications and is much more efficient than k-truss community search based approach.
- We develop a 2-level index structure that can handle various kinds of multiple-vertex k truss community search efficiently by first perform the k-truss community identity search on the top-level tree structured index and then, if necessary, search the exact k-truss community with the bottom-level index with a single run of breadth first search.
- We design an efficient bottom-up index construction algorithm for our 2-level index structure. The time and space complexity is $O(m \log m)$ and $O(m)$ respectively.
- We evaluate our index on large-scale real-world graphs and compare it with state-of-the-art algorithm. The results show that our index is not only much compact and efficient for k-truss community search queries but can also support various query types.

The rest of this paper is organized as follows. In Section 6 we show previous works on community search and detection. Section 2 provides notations and definitions used in this paper. We explain the two layered index structure for truss community search in Section 3. Section **??** discusses index update algorithm for dynamic graphs. The evaluations of our algorithm are in Section 5. We conclude our work in Section 7.

Note 1 add more application Durmaz et al (2017); Zong et al (2015); Yin and Shi (2017)

2 Preliminaries

In our problem, we consider an undirected, unweighted graph $G = (V, E)$. The number of vertices is denoted as $n = |V|$ and number of edges is denoted as $m = |E|$. If the graph is weighted, we use w_u and w_e to denote the weight of vertex u and edge e. We define the set of neighbors of a vertex v in G as $N_v = u \in V : (v, u) \in E$, and the degree of v as $d_v = |N_v|$. We define a triangle in G as a cycle of length 3. Let $u, v, w \in V$ be the three vertices on the cycle, and we denote this triangle by $\triangle_{uvw}$. Then we define several key concepts in this paper as follows.

Definition 1 (Edge support) *The support of an edge $e_{u,v} \in E$ is defined as $s_{e,G} = |\triangle_{uvw} : w \in V|$. We denote it as s_e when the context is clear.*

Definition 2 (Trussness) *The trussness of a subgraph $H \in G$ is the minimum support of edges in H plus 2, denoted by $\tau_H = min\{(s_{e,H}+2) : e \in E_H\}$. The trussness of an edge e is defined as: $\tau_e = max_{H \in G}\{\tau_H : e \in E_H\}$.*

Definition 3 (K-truss subgraph) *Given a graph G and $k \geq 2$, $H \subseteq G$ is a k-truss if $\forall e \in E_H, s_{e,H} \geq (k-2)$.*

Definition 4 (Maximal k-truss subgraph) *H is a maximal k-truss subgraph if it is not a subgraph of another k-truss subgraph with same trussness k in G.*

We use the same triangle adjacency and triangle connectivity definition as in Huang et al (2014) listed below.

Definition 5 (Triangle adjacency) *$\triangle_1$, $\triangle_2$ are adjacent if they share a common edge, i.e., $\triangle_1 \cap \triangle_2 \neq \emptyset$.*

Definition 6 (Triangle connectivity) *$\triangle_1, \triangle_2$ are triangle connected if they can reach each other through a series of adjacent triangles, i.e., for $1 \leq i < n, \triangle_i \cap \triangle_{i+1} \neq \emptyset$.*

Definition 7 (Triangle connected graph) *Two edge e_1, e_2 are triangle connected in a subgraph H if there are two triangle $\triangle_1, \triangle_2$ in H and $e_1 \in \triangle_1, e_2 \in \triangle_2$, either $\triangle_1 = \triangle_2$, or $\triangle_1$ is triangle connected with $\triangle_2$ in H. A graph G is triangle connected if all pairs of edges in G are triangle connected.*

Finally, we define k-truss community based on the definition of k-truss subgraph and triangle connectivity as follows.

Definition 8 (K-truss community) *A k-truss community is a maximal triangle connected k-truss subgraph.*

Figure 2 shows several examples of k-truss communities. The whole example graph is a 3-truss as every edge has support of at least 1. Note that there are 2 separate 4-truss communities in Figure 2 as they are not triangle connected with each other.

K-truss Community Search The problem of studied in this paper is defined as follows. Given a graph $G(V, E)$, a set of query vertices $Q \in V$, find all truss communities containing Q with maximum k, a specific k or any possible k.

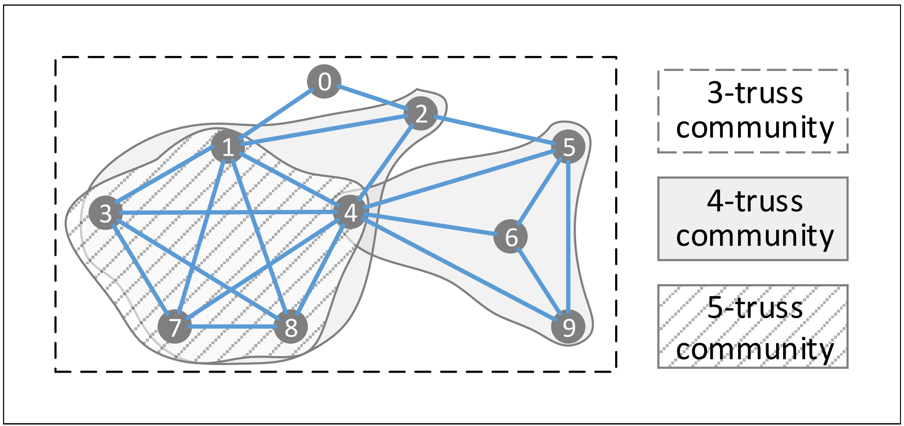


Fig. 2 An example graph for k-truss community

3 Indexed K-truss Community Search

We propose to solve k-truss community search problem using an index based approach. This section describes how to process a k-truss community search query on a static graph, including induced MST graph construction, creating tree-structured community graph, performing various kind of queries on the preprocessed index. In the next section, we describe index update procedure on dynamic graphs.

3.1 Induced MST Graph

[*** use counting sort *****]**

We first design an induced MST graph then propose the query algorithm based on it.

Induced MST Graph Construction. We first compute the edge trussness of graph G_o and then construct a new graph G_m, which we called induced MST graph, based on the graph G_o and its edges' trussness. We define the induced MST graph as follows.

Definition 9 (induced MST graph) *The induced MST graph is a weighted maximum spanning forest that each edge e in G_o is represented as a vertex x in G_m. An edge y in G_m represents that the two edges, which are represented by the two adjacent vertices of y, are contained in the same triangle in G_o. The weight of the each vertex in G_m is its represented edge's trussness in G_o. The weight of each edge in G_m is the lowest edge trussness of its related triangle's edges in G_o.*

We denote G'_m as the graph that is constructed the same way as G_m but with all triangles in G_o as edges, i.e., G_m is the maximum spanning forest of G'_m. We refer to lowest edge trussness of a triangle as the weight of the triangle.

We have the following theorem for vertex weights and edge weights in induced MST graph G_m.

Theorem 1 *In induced MST graph G_m, for each vertex x and each of its adjacent edge y, we have $w_x \geq w_y$.*

Proof According to Definition 9, w_x is the trussness of the represented edge e in G_o while w_y is the lowest trussness of edges in the represented triangle $\triangle$ in G_o. We have $\tau_e \geq \tau_\triangle$, therefore, $w_x \geq w_y$.

Algorithm 1: Induced MST graph construction

Data: $G_o(V_o, E_o)$, edge trussness $\{\tau_e, e \in E_o\}$
Result: $inducedMSTgraphG_m(V_m, E_m)$

$visited \leftarrow \emptyset$;
for $(u, v) \in E_o$ **do**
  suppose u is the lower degree end of (u, v);
  $V_m \leftarrow V_m \bigcup \{(u, v), \tau_{(u,v)}\}$;
  for $w \in N_u$ **do**
    if $(v, w) \in E_o$ ***and*** $\triangle_{uvw} \notin visited$ **then**
      $visited \leftarrow visited \bigcup \triangle_{uvw}$;
      $\tau_{\triangle_{uvw}} = min(\tau_{(u,v)}, \tau_{(u,w)}, \tau_{(v,w)})$;
      $V_m \leftarrow V_m \bigcup \{(u, w), \tau_{(u,w)}\}$;
      $V_m \leftarrow V_m \bigcup \{(v, w), \tau_{(v,w)}\}$;
      $E_m \leftarrow E_m \bigcup \{((u, v), (u, w)), \tau_{\triangle_{uvw}}\}$;
      $E_m \leftarrow E_m \bigcup \{((u, v), (v, w)), \tau_{\triangle_{uvw}}\}$;
      $E_m \leftarrow E_m \bigcup \{((u, w), (v, w)), \tau_{\triangle_{uvw}}\}$;
    end
  end
end
run Kruskal's algorithm on G_m;
return G_m

The truss decomposition algorithm Wang and Cheng (2012) is used to compute trussness of all edges $\{\tau_e, e \in E_o\}$ in G_o. Although it is possible to directly compute k-truss communities based on edge trussness with BFS traversals, such an algorithm suffers from high time complexity for redundant edge access Huang et al (2014). algorithm 1 uses both G_o and edge trussness as inputs to construct the induced MST graph G_m for optimal query time. The algorithm iterates through all edges of G_o and create a vertex in G_m for each edge (u, v) in G_o with weight $\tau_{(u,v)}$. Then for each unvisited neighbor triangle $\triangle_{uvw}$ of edge (u, v), the algorithm creates three edges $((u, v), (u, w))$, $((u, v), (v, w))$ and $((u, w), (v, w))$ in G_m with same weight $\tau_{\triangle_{uvw}} = min(\tau_{(u,v)}, \tau_{(u,w)}, \tau_{(v,w)})$. After that, one can simply run Kruskal's algorithm to get the maximum spanning forest.

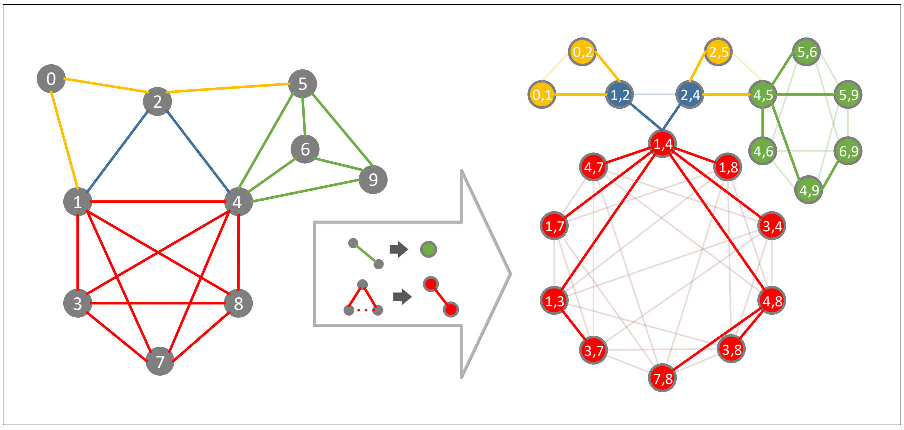


Fig. 3 An example induced MST graph of the example graph in Figure 2

We show an example of induced MST graph in Figure 3. We outline the induced MST graph of the example graph in Figure 2 with bold lines. The rest lines are edges that are generated by algorithm 1 but discarded by Kruskal's algorithm.

Note 2 some possible error of time complexity in Huang et al (2014) The time and space complexity for computation of edge trussness of G_o are $O(\sum_{(u,v)\in E_o} min\{d_u, d_v\})$ and $O(m)$ respectively Huang et al (2014). Listing all the triangles in G_o takes $O(\sum_{(u,v)\in E_o} min\{d_u, d_v\})$ time and $O(\sum_{(u,v)\in E_o} min\{d_u, d_v\})$ space. Finally, running Kruskal's algorithm takes $O(\sum_{(u,v)\in E_o} min\{d_u, d_v\}logm)$ time. As G_m is a maximum spanning forest, so the induced MST graph index takes $O(|V_m|) = O(m)$ space.

Query on Induced MST Graph. To query the k-truss communities of a query vertex q in G_o, the algorithm iterate through adjacent edges of the vertex q. For each neighbor edge (u, q) that is unvisited by the algorithm, it is marked as a seed edge for a new community C_i. Suppose the edge (u, q) in G_o is represented as a vertex x in G_m, the algorithm starts a BFS/DFS from vertex x in G_m and only expands through edges with weight $\geq k$ to find the connected component CC. Then if finds the represented edge e of each vertex $v \in CC$ and adds e to the community C_i. The union of all communities $A = \bigcup C_i$ is all the k-truss communities the vertex q belongs to.

Theorem 2 *The union of all communities $\bigcup C_i$ found by algorithm 2 is the union of all the k-truss communities containing query vertex q.*

Proof According to Definition 9, a vertex x in induced MST graph G_m with weight $w_x \leq k$ means the represented edge e in G_o has trussness $\tau_e \leq k$ and thus can be included in a k-truss community. An edge (x, y) in induced MST graph G_m with weight $w_{(x,y)} \leq k$ means the represented triangle $\triangle$ in G_o has all three edges with trussness higher or equal to k and thus the triangle is included in a k-truss community containing all three edges of it. Adjacent edges in G_m means adjacent triangles in G_o and connected components in G_m means triangle connected components in G_o. So, BFS/DFS search starts with a seed vertex x with weight constraint will find the maximal connected component including x which representing the k-truss community that e belongs to in G_o (x represents e in G_m). Therefore, performing such BFS/DFS searches on each edge of the query vertex will find all the k-truss communities that the query vertex belongs to.

Algorithm 2: Query on induced MST graph

Data: $G_o(V_o, E_o)$, $G_m(V_m, E_m)$, an integer k, a query vertex q
Result: a union of all k-truss communities $\bigcup C_i$ containing q

$i \leftarrow 0$, $visited \leftarrow \emptyset$;
for $u \in N_q$ **do**
  if $(u, q) \notin visited$ **then**
    find representing vertex x of (u, q) in G_m;
    $CC \leftarrow$ connected component containing x with edges of weight $\geq k$;
    $C_i \leftarrow \emptyset$;
    for $v \in CC$ **do**
      find represented e of v in G_o;
      $visited \leftarrow visited \bigcup e$;
      $C_i \leftarrow C_i \bigcup e$;
    end
    $i \leftarrow i + 1$;
  end
end
return $\bigcup C_i$

Since the query process is performing a BFS on a maximum spanning forest, each query takes $O(|A|)$ time and $O(|A|)$ space, where $|A|$ is the number of edges in A. Although such time complexity is already optimal if the detailed communities are required. We propose a new index structure that can be constructed upon the induced MST graph to further reduce the time complexity if details of k-truss communities are not required.

3.2 Tree-structured Community Graph

We first show how to construct the tree-structured community graph based on induced MST graph. Then we design an algorithm to efficiently query the tree-structured community graph.

Tree-structured Community Graph Construction. A key observation in Cohen (2008) is that, for $k \geq 2$, each k-truss of G_o is the subgraph of a (k-1)-truss of G_o. With this observation, for k-truss communities, we have the following theorem.

Theorem 3 *A k-truss community C_k is the subgraph of a l-truss community C_l, if C_k and C_l are triangle connected and $l < k$. If k-truss community C_k is the subgraph of both l_1-truss community C_{l_1} and l_2-truss community C_{l_2}, then $l_1 \neq l_2$.*

Proof For the first part, since $l < k$, if edges in C_k are triangle connected through triangles with trussness of k, then they are also triangle connected through triangles with trussness of l.

Note 3 do we call it k-truss or l_1-truss. For the second part, suppose $l_1 = l_2$, then edges in C_{l_1} and C_{l_2} are triangle connected through C_k. So $C_{l_1} \bigcup C_{l_2}$ meets the definition of k-truss community (Definition 8) and becomes a larger k-truss community. This contradicts with C_{l_1} and C_{l_2} are k-truss communities themselves, i.e., they are maximal k-truss.

According to Theorem 3, we can build another tree-structured index upon our existing induced MST graph to further facilitate KTruss computation. In this new tree-structured index, we use vertices to represent k-truss communities, i.e., we assign each k-truss community an unique ID and a representing vertex in the new index. If one k-truss community is the subgraqh of another k-truss community, we assign an edge to connect the representing vertices. Each vertex can have a list associated with it including the status of the related k-truss community, such as the trussness of the community, the size of the community, etc. We call this new index the tree-structured community graph and denote it as G_t. For each vetx of G_t, we also have meta data of the represented k-truss communities, e.g., the trussness, the size, etc., stored with it. These meta-data can be gathered very easily through the index construction process. For the ease of query, we build a hash table h that for each edge e in G_o (vertex x in G_m), we record the ID of the k-truss community that includes it with highest order k. We denote such a k-truss community as C^{max}. We have the following theorem for G_t.

Theorem 4 *The tree-structured community graph G_t is a forest.*

Proof First, according to Theorem 3, there is only one ancestor for each k-truss community for . Also, there is no inter level edges according to the definition of maximal KTruss. So, if the graph contains a loop, then a KTruss may contains more than 1 ancestors.

Second, G_t can be disconnected as not all k-truss communities are triangle connected with each other.

algorithm 3 shows the procedure to build the tree-structured community graph G_t. The algorithm uses BFS to traverse the induced MST graphG_m. For each vertex x, if it does not have a parent vertex in the BFS traversal, then the algorithm uses it as a seed vertex to create a new index tree. Otherwise it is combined to the same index tree $T \in G_t$ as its parent vertex y. According to Theorem 1, we have the following equation.

Algorithm 3: Tree-structured community graph Construction

Data: $G_m(V_m, E_m)$
Result: $G_t(V_t, E_t)$, h

$Q \leftarrow \emptyset$, $parent \leftarrow \emptyset$;
while $V_m \neq \emptyset$ **do**
    $seed \leftarrow$ an unvisited vertex in V_m, $Q \leftarrow Q \bigcup seed$;
    while $Q \neq \emptyset$ **do**
        $x = Q.pop()$;
        for $z \in N_x$ **do**
            $Q \leftarrow Q \bigcup z$, $parent[z] \leftarrow x$;
        end
        if $x \in parent$ **then**
            $y \leftarrow parent[x]$, $C_a \leftarrow C_y^{max}$;
            while $\tau_{C_a} > w_{(x,y)}$ **do**
                $C_c \leftarrow C_a$, $C_a \leftarrow$ parent of C_a in G_t;
                if $C_a = \emptyset$ **then**
                    $\tau_{C_a} \leftarrow -1$ ▷ Reach the top of the tree.;
                end
            end
            if $\tau_{C_a} < w_{(x,y)}$ **then**
                if $w_{(x,y)} = w_x$ **then**
                    create C_x^{max}, $h[x] \leftarrow C_x^{max}$;
                    $C_x^{max}.parent \leftarrow C_a$, $C_c.parent \leftarrow C_x^{max}$;
                else
                    create C_x^{max}, $h[x] \leftarrow C_x^{max}$;
                    create $C_{(x,y)}$, $C_{(x,y)}.parent \leftarrow C_a$;
                    $C_c.parent \leftarrow C_{(x,y)}$;
                    $C_x^{max}.parent \leftarrow C_{(x,y)}$;
                end
            else
                if $w_{(x,y)} = w_x$ **then**
                    $h[x] \leftarrow C_a$;
                else
                    create C_x^{max}, $h[x] \leftarrow C_x^{max}$;
                    $C_x^{max}.parent \leftarrow C_a$;
                end
            end
        else
            create C_x^{max}, $h[x] \leftarrow C_x^{max}$;
            $V_t \leftarrow V_t \bigcup C_x^{max}$;
        end
        remove x from V_m;
    end
end
return $G_t(V_t, E_t)$, h

$$w_x \geq w_{(x,y)}, w_y \geq w_{(x,y)} \tag{1}$$

An example is shown in Figure 4

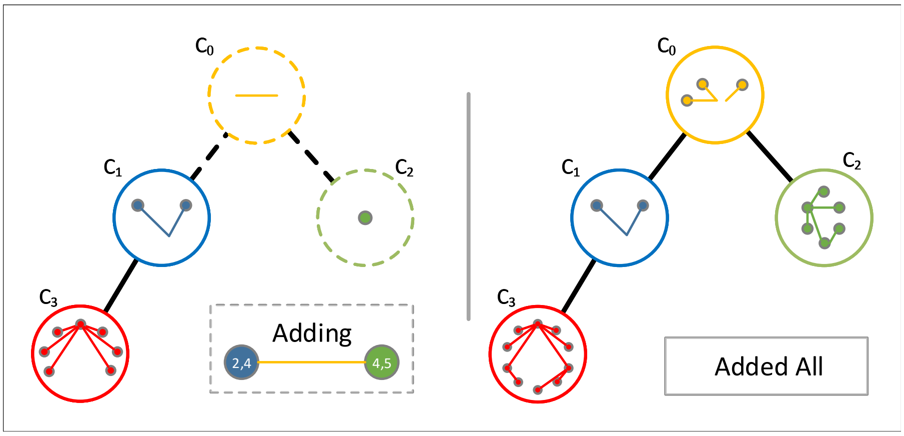


Fig. 4 An example tree-structured community graph of the induced MST graph in Figure 3

Theorem 5 *For a vertex x and its neighbor vertex y in induced MST graph G_m, if their representing edges in G_o are contained in the same k-truss community with trussness of k, then $k \leq w_{(x,y)}$.*

Proof Since G_m is the maximum spanning forest, it has the cycle property, i.e., for any cycle in G'_m, if the weight of an edge in the cycle is smaller than the individual weights of all the other edges in the cycle, then this edge cannot belong to a maximum spanning forest. So there is no path in G'_m between x and y that has all edges with weight $> w_{(x,y)}$. Suppose x and y representing e_x and e_y in G_o, this means that e_x is not triangle connected to e_y through edges in G_o with trussness $> w_{(x,y)}$. Therefore, it is not possible for e_x and e_y to exist in the same k-truss community with $k > w_{(x,y)}$.

Having a parent y in the BFS search only means that the vertex x can be combined to the current index tree T. We still have a problem to solve: On which part of T should the algorithm add the vertex x? According to Theorem 5 and Equation 1, the algorithm needs to backtrack T from C_y^{max} to find an ancestor vertex C_a that meets $\tau_{C_a} \leq w_{(x,y)}$ and use it as the merge point of x. We refer to the index vertices $C_y^{max}, ..., C_i, ..., C_a$ as the backtrack branch for vertex x in T and denote it as B.

Once the algorithm has found C_a, it needs to check the relations of τ_{C_a}, $w_{(x,y)}$ and w_x to decide how to merge vertex x to T. Note that they follow $\tau_{C_a} \leq w_{(x,y)} \leq w_x$, so we have 4 cases shown in algorithm 3. As long as $\tau_{C_a} \neq w_x$, we create a new index vertex C_x^{max} with trussness $\tau_{C_x^{max}} = w_x$. If $\tau_{C_a} < w_{(x,y)} < w_x$, we also create a new index vertex $C_{(x,y)}$ with trussness $\tau_{C_{(x,y)}} = w_{(x,y)}$. Then we adjust the tree structure of T with new index vertices. Finally, we update the hash table to record in which index vertex x is.

For each vertex of G_m, the backtrack procedure takes $O(k_{max})$ time, where k_{max} is the highest trussness of any k-truss community in G_o. Since the index construction process is a BFS on a maximum spanning tree, the tree-structured community graph construction algorithm takes $O(k_{max}m)$ time. As each vertex in G_t represents a k-truss community in G_o, and G_t is a forest. The algorithm takes $O(m)$ space and the index size is also $O(m)$ space. Although in practice, the size of G_t is much smaller than $O(m)$.

4 K-truss community search query

Query on Tree-structured Community Graph. Tree-structured community graph supports three basic types of k-truss community queries of a single query vertex q as listed below.

- K-truss query: Given a vertex q and an integer k, find the k-truss community that contains q.
- Max-k-truss query: Given a vertex q, find the k-truss community with highest possible trussness that contains q.
- Any-k-truss query: Given a vertex q, find all the k-truss communities that contains q.

Max-k-truss query is naturally supported by simply looking up the hash table h and comparing trussness of $h[x_e]$ for each neighbor edge. We show the queries process algorithms for k-truss query and any-k-truss query in algorithm 4. A common operation used in both query algorithms is what we called backtrack branch search, which is defined in Definition 10 below. We can see that if a specific k is provided, the backtrack branch search will stop once the trussness falls below k. On the other hand, if no k is provided, a value of 0 is used and the search will reach the root of the tree.

Definition 10 (Backtrack branch search) *Given a vertex $C_0 \in G_t$ and an integer k, the backtrack branch search returns a list of vertices $C_0, ..., C_i, ...$ that C_{i+1} is the parent vertex of C_i in G_t and any vertex C_i meets $\tau_{C_i} \geq k$. We refer to the searching results $C_0, ..., C_i, ...$ as backtrack branch and denote it as B.*

Tree-structured community graph also supports all three types of queries when the input is a set of query vertices Q. The query process algorithms simply takes intersections of the query results of each individual query vertex for k-truss queries and any-k-truss queries. For max-k-truss queries, the query process algorithm needs to calculate the least common ancestors in G_t of the results of each individual query vertex.

For single vertex queries, the time complexity is $O(d_q)$ for max-k-truss queries and $O(\sum_{e\in N_q} \tau_{h[x_e]})$ for k-truss and any-k-truss queries. The space complexity is $O(1)$ for max-k-truss queries, $O(d_q)$ for k-truss queries and $\sum_{e\in N_q} \tau_{h[x_e]}$ for any-k-truss queries. For multiple vertices max-k-ktruss queries, since the least common ancestor computation takes $O(H)$[1] time, where H is

[1] $O(H)$ is for simple online algorithm, off-line algorithms can achieve time complexity of $O(1)$ Bender and Farach-Colton (2000).

Algorithm 4: Query on Tree-structured community graph

Data: $G_o(V_o, E_o)$, $G_t(V_t, E_t)$, the hash table h, a query vertex q or a set of query vertices Q, [an integer k]
Result: a set of k-truss community IDs R

```
function branch_search (C ∈ G_t, G_t, [k = 0])
    B ← ∅;
    while C ≠ ∅ and τ_C ≥ k do
        B ← B ⋃ C;
        C ← C.parent;
    end
    return B
end
function query_k (q, G_o, G_t, k)
    R ← ∅;
    for e ∈ N_q do
        B ← branch_search (h[x_e], G_t, k);
        if τ_{B[-1]} = k then
            R ← R ⋃ B[-1];                ▷ B[-1] is the last element in B
        end
    end
    return R
end
function query_anyk (q, G_o, G_t)
    R ← ∅;
    for e ∈ N_q do
        B ← branch_search (h[x_e], G_t);
        R ← R ⋃ B;
    end
    return R
end
```

the height of the tree. The query time is $O(\sum_{q\in Q}(\max_{e\in N_q} \tau_{h[x_e]} + d_q))$ and the space is $O(\max_{q\in Q} \max_{e\in N_q} \tau_{h[x_e]})$. Multiple vertices k-truss queries take $O(\sum_{q\in Q} \sum_{e\in N_q} \tau_{h[x_e]})$ time and $O(\max_{q\in Q} d_q)$ space. For multiple vertices any-k-truss queries, the time and space complexity is $O(\sum_{q\in Q} \sum_{e\in N_q} \tau_{h[x_e]})$ and $O(\max_{q\in Q} \sum_{e\in N_q} \tau_{h[x_e]})$ respectively.

5 Evaluations

In this section, we evaluate our proposed index structure for k-truss community queries on real-world networks.

Datasets

We evaluate our algorithm on 5 graphs from different disciplines as shown in table 1. To simplify our experiments, we treat them as undirected, un-weighted graphs and only use the largest weakly connected component of each graph. All datasets are collected from Stanford Network Analysis Project Leskovec and Krevl (2014) and Network Repository Rossi and Ahmed (2015).

Table 1 Datasets

Dataset	Type	$\lvert V_{wcc}\rvert$	$\lvert E_{wcc}\rvert$	$\lvert\triangle\rvert$	k_{max}
Wiki	Communication	2.4M	4.7M		
Skitter	Internet	1.7M	11.1M		
Livejournal	Social	4.8M	43.4M		
Orkut	Social	3M	117M		
Sinaweibo	Social	58.7M	261.3M		

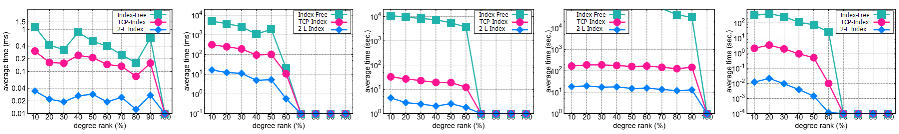
Datasets with the number of vertices and edges in the largest weakly connected components, the number of triangles and the maximum trussness of the graph.

Experiment settings

We evaluate our algorithms, we use a Cloudlab Ricci et al (2014) c8220 server with two 10-core 2.2GHz E5-2660 processors and 256GB memory. All algorithms are implemented in C++.

5.1 Query time

We evaluate the query time of various types of k-truss community queries on our index structure. We first evaluate the single vertex k-truss community search and compare the query time with the TCP index proposed in Huang et al (2014). We also use index free scheme as a baseline. As the k-truss community search time heavily rely on the degree of the query vertex, so we follow the similar procedure used in Huang et al (2014). We partition the vertices in each graph according to their degree into 10 categories and at each category, we randomly select 100 vertices to perform 100 independent k-truss community search. For all the queries, we fix the k at 10, which results in communities sufficiently large when degree of the query vertex is high to show the performance difference of different algorithms. For each query vertex, we first perform k-truss community identity search to find identity and seed edge for each resulting community on top-level index, and then we perform the k-truss community search on bottom-level index with breath first search.



(a) Wiki (b) Skitter (c) LiveJournal (d) Orkut (e) Sina Weibo

Fig. 5 Single vertex query for exact truss community search.

We show the results for single vertex k-truss community search in figure 5. The results show that our index outperforms the TCP index proposed in Huang et al (2014) by at least an order of magnitude. And both index

Table 2 Comparison of Index Construction

Dataset	Graph Size	Index Size		Index Time	
		TCP	Our	TCP	Our
Wiki	57	296	187	138	117
Skitter	149	485	430	873	682
Livejournal	635	3174	2699	1686	1557
Orkut	1769	8174	5742	3342	2884
Sinaweibo	4050	9322	7689	7730	6938

scheme outperform the index-free method by sevearl orders of magnitude. Because the algorithm get rid of costly triangle connectivity discovery at query time and use straight forward edge connectivity by performing breath first search.

5.2 Index construction time and size

We show in this section the index size and index construction time of our scheme compared to TCP-index in table 2. Both indices are generated in memory and we show the size of the data structures that hold the index. We exclude the truss decomposition time for both scheme so that the index construction time only shows how long it takes to generate a certain index with edge trussness provided. We can see in table 2 that our scheme have smaller index size for most graphs and takes shorter time to generate the index compared to the TCP-index. Note that the index size contains both level of index structures. If only the k-truss community identity search is required, then the index size required for performing online queries is much smaller than the number provided here.

6 Related Works

Our work is most related to the inspiring work Huang et al (2014) which introduce the notion of k-truss community based on triangle connectivity. An index structure call TCP is proposed in Huang et al (2014) that each vertex holds their maximum spanning forest based on edge trussness of their ego-network. Triangle connected k-truss communities mitigate the "free-rider" issue but at the cost of slow computation efficiency especially for vertices belongs to large k-truss communities so that they are not able to meet requirements of queries in some cases. We have use this work as a comparison in section section 5.

The notion of triangle connected k-truss community is also referred to as $k-(2,3)$ nucleus in Sariyüce and Pinar (2016) where they propose an approach based on disjoint-set forest to speed up the process of nuclues decomposition. We are more emphasize on the fast query process rather speed up the truss decomposition. Also the tree-structured community graph is easier to maintain for dynamic graphs.

Our work falls in the category of cohesive subgraph mining Huang et al (2017); Koujaku et al (2016); Sozio and Gionis (2010); Cui et al (2014); Li et al (2015); Cui et al (2013); McAuley and Leskovec (2012), such as clique Bron and Kerbosch (1973); Rossi et al (2014), k-core Cheng et al (2011); Shin et al (2016); Barbieri et al (2015); Li et al (2017c), k-truss Huang et al (2014); Wang and Cheng (2012); Cohen (2008); Huang et al (2015, 2016); Zheng et al (2017), k-plex Wang et al (????) and quasi-clique Tsourakakis et al (2013); Lee and Lakshmanan (2016). An α-adjacency γ-quasi-k-clique model is introduced by Cui et al (2014) for online searching of overlapping communities. ρ-dense core is a pseudo clique recently introduced by Koujaku et al (2016) that is able to deliver optimal solution for graph partition problems. The pattern of k-core structures is studied by Shin et al (2016) for applications such as finding anomalies in real world graphs, approximate degeneracy of large-scale graphs and so on. K-truss decomposition is also studied in Huang et al (2016); Zou and Zhu (2017) for probabilistic graphs.

Li et al (2015) introduce a novel community model called k-influential community based on the concept of k-core, which can capture the influence of a community. Chen et al (2016); **?**); Li et al (2017b); Bi et al (2017); Li et al (2017a) also study this.

Wu et al (2015) systematically study the existing goodness metrics and provide theoretical explanations on why they may cause the free rider effect. We further develop a query biased node weighting scheme to reduce the free rider effect.

Huang et al (2015) tries to address the "free rider" issue by finding communities that meet cohesive criteria with the minimum diameter.

Fang et al (2016) works on attribute graph, finding communities that satisfies both structure cohesiveness (i.e., its vertices are tightly connected) and keyword cohesiveness (i.e., its vertices share common keywords). Huang and Lakshmanan (2016); Shang et al (2016, 2017); Zhang et al (2017); Fang et al (2017); Huang and Lakshmanan (2017) also works on this.

Sariyüce et al (2017) nucleus decompositions

7 Conclusion

In this paper, we describe ...

References

Barbieri N, Bonchi F, Galimberti E, Gullo F (2015) Efficient and effective community search. Data Mining and Knowledge Discovery 29(5):1406–1433

Bender MA, Farach-Colton M (2000) The lca problem revisited. In: LATIN 2000: Theoretical Informatics, Springer

Bi F, Chang L, Lin X, Zhang W (2017) An optimal and progressive approach to online search of top-k important communities. arXiv preprint arXiv:171105857

Bron C, Kerbosch J (1973) Algorithm 457: finding all cliques of an undirected graph. Communications of the ACM 16(9):575–577

Chen S, Wei R, Popova D, Thomo A (2016) Efficient computation of importance based communities in web-scale networks using a single machine. In: Proceedings of the 25th ACM International on Conference on Information and Knowledge Management, ACM, pp 1553–1562

Cheng J, Ke Y, Chu S, Özsu MT (2011) Efficient core decomposition in massive networks. In: Data Engineering (ICDE), 2011 IEEE 27th International Conference on, IEEE, pp 51–62

Cohen J (2008) Trusses: Cohesive subgraphs for social network analysis. National Security Agency Technical Report 16

Cui W, Xiao Y, Wang H, Lu Y, Wang W (2013) Online search of overlapping communities. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, ACM, pp 277–288

Cui W, Xiao Y, Wang H, Wang W (2014) Local search of communities in large graphs. In: Proceedings of the 2014 ACM SIGMOD international conference on Management of data, ACM, pp 991–1002

Durmaz A, Henderson TA, Brubaker D, Bebek G (2017) Frequent subgraph mining of personalized signaling pathway networks groups patients with frequently dysregulated disease pathways and predicts prognosis. In: PSB

Fang Y, Cheng R, Luo S, Hu J (2016) Effective community search for large attributed graphs. Proceedings of the VLDB Endowment 9(12):1233–1244

Fang Y, Cheng R, Chen Y, Luo S, Hu J (2017) Effective and efficient attributed community search. The VLDB Journal pp 1–26

Huang X, Lakshmanan LV (2016) Attribute truss community search. arXiv preprint arXiv:160900090

Huang X, Lakshmanan LV (2017) Attribute-driven community search. Proceedings of the VLDB Endowment 10(9):949–960

Huang X, Cheng H, Qin L, Tian W, Yu JX (2014) Querying k-truss community in large and dynamic graphs. In: Proceedings of the 2014 ACM SIGMOD international conference on Management of data, ACM, pp 1311–1322

Huang X, Lakshmanan LV, Yu JX, Cheng H (2015) Approximate closest community search in networks. Proceedings of the VLDB Endowment 9(4):276–287

Huang X, Lu W, Lakshmanan LV (2016) Truss decomposition of probabilistic graphs: Semantics and algorithms. In: Proceedings of the 2016 International Conference on Management of Data, ACM, pp 77–90

Huang X, Lakshmanan LV, Xu J (2017) Community search over big graphs: Models, algorithms, and opportunities. In: Data Engineering (ICDE), 2017 IEEE 33rd International Conference on, IEEE, pp 1451–1454

Koujaku S, Takigawa I, Kudo M, Imai H (2016) Dense core model for cohesive subgraph discovery. Social Networks 44:143–152

Lee P, Lakshmanan LV (2016) Query-driven maximum quasi-clique search. In: Proceedings of the 2016 SIAM International Conference on Data Mining, SIAM, pp 522–530

Leskovec J, Krevl A (2014) SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data

Li J, Wang X, Deng K, Yang X, Sellis T, Yu JX (2017a) Most influential community search over large social networks. In: Data Engineering (ICDE), 2017 IEEE 33rd International Conference on, IEEE, pp 871–882

Li RH, Qin L, Yu JX, Mao R (2015) Influential community search in large networks. Proceedings of the VLDB Endowment 8(5):509–520

Li RH, Qin L, Yu JX, Mao R (2017b) Finding influential communities in massive networks. The VLDB Journal pp 1–26

Li Zj, Zhang WP, Li RH, Guo J, Huang X, Mao R (2017c) Discovering hierarchical subgraphs of k-core-truss. In: International Conference on Web Information Systems Engineering, Springer, pp 441–456

McAuley JJ, Leskovec J (2012) Learning to discover social circles in ego networks. In: NIPS, vol 2012, pp 548–56

Ricci R, Eide E, The CloudLab Team (2014) Introducing CloudLab: Scientific infrastructure for advancing cloud architectures and applications. USENIX ;login: 39(6), URL https://www.usenix.org/publications/login/dec14/ricci

Rossi RA, Ahmed NK (2015) The network data repository with interactive graph analytics and visualization. In: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, URL http://networkrepository.com

Rossi RA, Gleich DF, Gebremedhin AH, Patwary MMA (2014) Fast maximum clique algorithms for large graphs. In: Proceedings of the 23rd International Conference on World Wide Web, ACM, pp 365–366

Sariyüce AE, Pinar A (2016) Fast hierarchy construction for dense subgraphs. Proceedings of the VLDB Endowment 10(3):97–108

Sariyüce AE, Seshadhri C, Pinar A, Çatalyürek ÜV (2017) Nucleus decompositions for identifying hierarchy of dense subgraphs. ACM Transactions on the Web (TWEB) 11(3):16

Shang J, Wang C, Wang C, Guo G, Qian J (2016) Agar: an attribute-based graph refining method for community search. In: Proceedings of the Sixth International Conference on Emerging Databases: Technologies, Applications, and Theory, ACM, pp 65–66

Shang J, Wang C, Wang C, Guo G, Qian J (2017) An attribute-based community search method with graph refining. The Journal of Supercomputing pp 1–28

Shin K, Eliassi-Rad T, Faloutsos C (2016) Corescope: Graph mining using k-core analysis-patterns, anomalies and algorithms. In: Data Mining (ICDM), 2016 IEEE 16th International Conference on, IEEE, pp 469–478

Sozio M, Gionis A (2010) The community-search problem and how to plan a successful cocktail party. In: Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, pp 939–948

Tsourakakis C, Bonchi F, Gionis A, Gullo F, Tsiarli M (2013) Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees. In: Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, pp 104–112

Wang J, Cheng J (2012) Truss decomposition in massive networks. Proceedings of the VLDB Endowment 5(9):812–823

Wang Y, Xun J, Yang Z, Li J (????) Query optimal k-plex based community in graphs. Data Science and Engineering pp 1–17

Wu Y, Jin R, Li J, Zhang X (2015) Robust local community detection: on free rider effect and its elimination. Proceedings of the VLDB Endowment 8(7):798–809

Yin Z, Shi X (2017) Taming near repeat calculation for crime analysis via cohesive subgraph computing. arXiv preprint arXiv:170507746

Zhang Y, Wang Y, Parthasarathy S (2017) Visualizing attributed graphs via terrain metaphor. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, pp 1325–1334

Zheng Z, Ye F, Li RH, Ling G, Jin T (2017) Finding weighted k-truss communities in large networks. Information Sciences 417:344–360

Zong B, Xiao X, Li Z, Wu Z, Qian Z, Yan X, Singh AK, Jiang G (2015) Behavior query discovery in system-generated temporal graphs. Proceedings of the VLDB Endowment 9(4):240–251

Zou Z, Zhu R (2017) Truss decomposition of uncertain graphs. Knowledge and Information Systems 50(1):197–230