2021 IEEE 37th International Conference on Data Engineering (ICDE)

2021 IEEE 37th International Conference on Data Engineering (ICDE) | 978-1-7281-9184-3/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICDE51399.2021.00016

# Efficient Community Search with Size Constraint

Boge Liu[‡†], Fan Zhang[‡], Wenjie Zhang[†], Xuemin Lin[†], Ying Zhang[§]

[‡]*Guangzhou University*, [†]*University of New South Wales*, [§]*University of Technology Sydney*
boge.liu@unsw.edu.au, zhangf@gzhu.edu.cn, {lxue, zhangw}@cse.unsw.edu.au, ying.zhang@uts.edu.au

***Abstract*—The studies of $k$-truss based community search demonstrated that it can find high-quality personalized communities with good properties such as high connectivity and bounded diameter. Motivated by natural restrictions from real applications, in this paper, we investigate the search of triangle-connected $k$-truss with size constraint (denoted by SCkT) in a graph $G$: given a size constraint $s$, an integer $k$, and query set $Q$, SCkT search aims to find a triangle-connected $k$-truss $H$ containing the vertices in $Q$ and with size (i.e., total number of vertices in $H$) not exceeding $s$. We prove that the SCkT search problem is NP-hard. To tame the hardness, we fully exploit the properties of triangle-connected $k$-truss subgraphs s.t. a practically-efficient exact solution for SCkT search is developed. A novel and effective lower bound is proposed to early terminate unpromising search branches and narrow down the search space. Two search strategies, *expansion* and *shrinking*, are investigated to tailor for efficient support of SCkT search. A hybrid search method is proposed combining the expansion and shrinking strategies, where a score function is used to guide the search order. Our extensive experiments on real-life and synthetic graphs demonstrate the effectiveness of the SCkT model and the efficiency of the proposed techniques.**

## I. Introduction

Graphs are widely used to model the relationships of entities in a large spectrum of applications including social networks, World Wide Web, collaboration networks and biological networks. Query processing and mining with communities is one of the fundamental problems in graph analytics, which extracts densely connected structures from large graphs. In many real-life scenarios, users are more interested in the communities they participate in. Thus, community search, which aims to find communities containing given query vertices, is widely studied [17]. It enables personalized community discovery to fulfill the requirements from query users, and has thus found a wide range of applications, e.g., team formation [49], social contagion modeling [37], and identification of protein functions [14].

In this paper, we follow the study of $k$-truss based community search [1], [20] for its effectiveness in finding high-quality communities with promising properties. Given a graph $G$, a $k$-truss of $G$ is the maximal subgraph in which every edge is contained in at least $k-2$ triangles in the subgraph. In the literature, different models are proposed to measure community cohesiveness, such as $k$-core [46], [47], $k$-truss [11], [20], and clique [31]. The $k$-truss is an elegant relaxation of the over-restricted clique model, and also an enhanced version of the $k$-core model. The $k$-core is considered as the seedbed for finding denser subgraphs [46], [47]. Note that each vertex in the $k$-truss has at least $k-1$ neighbors inside, i.e., the $k$-truss is always a (dense) subgraph of ($k$-1)-core. Instead of primitive nodes or edges, $k$-truss exploits the higher-order graph motif, triangle, to capture the cohesiveness in communities with strong theoretical guarantees [6], [42], [43] (more details are given in Section II). The $k$-truss also has its advantage in computation cost: finding all $k$-trusses from a graph consumes polynomial time [38].

In a social network, the users in triangles usually have strong and stable relationship since it implies these users having some common friends [6], [42], [43]. On top of the $k$-truss model, Huang et al. [20] impose a triangle connectivity constraint that requires two edges in the same community either belong to the same triangle, or are reachable through a series of adjacent triangles. This triangle connectivity is more rigorous than primitive connectivity which only requires two vertices are reachable from each other through a simple path. Due to the effectiveness of triangles in modeling the community cohesiveness, [1], [20] adopt the $k$-truss model augmented with triangle connectivity in community search.

While it has been shown that $k$-truss model and triangle-connected $k$-truss model work well in finding communities containing query vertices [1], [20], [38], existing studies mainly focus on the *maximal* (triangle-connected) $k$-truss computation (e.g., [1], [20], [21], [50]) which aims to find the largest induced (triangle-connected) $k$-truss subgraph. However, in many real scenarios especially one or a set of query vertices are involved, there tends to be a natural constraint over the number of vertices in a community due to the limitation of capability or budget [4], [36]. Based on this fact, the size constraint is considered in some community search problems, to find communities with size no larger than a given threshold, e.g., [17], [36].

In this paper, we study the problem of searching triangle-connected $k$-truss community with size constraint (SCkT). Given a set of query vertices, SCkT search aims to find a triangle-connected $k$-truss containing all the query vertices with size no larger than a given threshold $s$. In Figure 1, the whole graph is a triangle-connected $k$-truss with $k=4$. Suppose $v_1$ is the query vertex, $k=4$ and $s=6$, the search of SCkT will return the shaded subgraph in Figure 1 induced by the vertices $v_1$ to $v_6$.

We illustrate the importance of SCkT with the following representative applications:

[*] Boge Liu and Fan Zhang are the joint first authors. Fan Zhang is the corresponding author.

978-1-7281-9184-3/21/$31.00 ©2021 IEEE
DOI 10.1109/ICDE51399.2021.00016

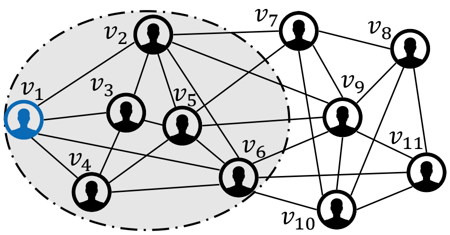


Fig. 1. Motivating Example with $k = 4$

*(1) Team Formation.* In online social platforms (e.g., Meetup, Facebook), a host may wish to organize a group activity such as hiking, sport, or party. Besides social connections, a size constraint is often applied in assembling the group due to various factors such as availability of tickets or capacity of the venue [17], [36]. In this scenario, a certain number of friends in the social platform will be recommended to the host by the SCkT search.

*(2) Biological Network Analysis.* In biology, a fundamental task is to detect and identify functional modules, which technically are cohesive subgraphs in protein-protein interaction networks [3], [14], [35]. The proteins in a cohesive subgraph tend to have similar functions. As shown in [23], triangle-connected $k$-truss model can be used to identify proteins with similar functions. A recent study shows such homogeneous property mainly holds for small cohesive subgraphs [24].

*(3) Suspicious Group Detection.* In transaction networks (e.g. Alipay) where users are connected by transfers, it is important to identify suspicious groups which often correspond to cohesive subgraphs. However, mining a subgraph with large size would include many innocent users s.t. a large manual cost for check is needed. Therefore, given a user in the black list, it is more desirable to find a cohesive subgraph containing the user with size constraint.

**Challenges.** To the best of our knowledge, this paper is the first work to study the problem of SCkT search. We prove that the SCkT search problem is NP-hard. [23] and [1] compute all the maximal triangle-connected $k$-trusss offline and build indexes based on the offline results to support online queries. However, their indexes cannot be applied to SCkT as they only record the maximal triangle-connected $k$-truss for each vertex. If we enforce the use of their indexes for SCkT search, we have to dynamically maintain the indexes as the size constraint requires numerous edge insertion/deletion in the search, which is prohibitively expensive. Nevertheless, we can utilize their indexes in finding the initial triangle-connected $k$-truss that contains query vertices. It is thus essential to develop techniques which judiciously determine the search branch and narrow down the search space.

**Our Solution.** To overcome above challenges, we first develop a novel size lower bound of triangle-connected $k$-truss. In the search of SCkT, if the lower bound of a partial solution is lager than the size threshold, we can safely early terminate this unpromising search branch. To quickly find a feasible solution, we propose an expansion-based search strategy, in which we start from the query vertex $q$ and iteratively expand with the most promising vertex to form a SCkT. To quantify whether a vertex is promising or not, we propose a score function based on the distribution of its neighbors in current partial solution. Besides, we observe that removing one vertex in a triangle-connected $k$-truss may lead to the removal of many other vertices. Thus, we also propose the shrinking-based search strategy, in which we start from the entire triangle-connected $k$-truss and keep removing the most unpromising vertex until the remaining graph forms a SCkT. Finally, we design a hybrid search combining the expansion and shrinking strategies, which automatically guides the search order. As shown in our experiment, the hybrid search can fast return almost all the queries.

**Contributions.** The principal contributions of the paper are summarized as follows.

- Motivated by real-life requirements, we apply the size constraint to the triangle-connected $k$-truss community model. The paper is the first to study the problem of size-constrained triangle-connected $k$-truss community (SCkT) search. We prove that the SCkT search problem is NP-hard.
- We develop a practically-efficient exact solution for SCkT search. A novel and effective lower bound is proposed to early terminate unpromising search branches to enhance efficiency.
- Two search strategies, expansion and shrinking, are investigated to provide efficient support for SCkT search regarding different situations of current partial solution. Through a careful combination of the expansion and shrinking, we propose a hybrid search method to return triangle-connected $k$-truss communities with high quality.
- Extensive experiments on real graphs and synthetic graphs are conducted to evaluate the proposed techniques. The results demonstrate the efficiency and effectiveness of our methods. In particular, over the largest graph with over 41 million vertices and over 1.5 billion edges, our algorithms can retrieve a SCkT within 10s for over 95% of the queries.

## II. Related Work

Mining cohesive subgraphs is a fundamental problem in graph analysis, which reveals potential community structures of real-world graphs. In the literature, numerous models have been proposed to quantify cohesive subgraphs including $k$-clique [44], $k$-core [13], [32], [36], [39], $k$-truss [11], [20], $k$-ECC [8], [19], and nucleus [33], [34]. There are also many extensions to tradition models, such as $(\alpha, \beta)$-core [25], [41], $(k, p)$-core [45], CoreCube [27], etc. We review two research areas closely related to the SCkT community search.

**$k$-Truss.** The $k$-truss model is defined on the higher-order graph motif, triangle, and has numerous nice properties in community modeling and computation [11]. For example, a $k$-truss is a ($k$-1)-core and a ($k$-1)-ECC but not vice versa [11], which means that each vertex in $k$-truss has at least $k - 1$ neighbors and any deletion of fewer than $k - 1$ edges cannot disconnect a $k$-truss. Moreover, $k$-truss is diameter-bounded [22], i.e., a $k$-truss with $n$ vertices has a diameter within $\lfloor \frac{2n-2}{k} \rfloor$. Due to its nice properties, $k$-truss has been wildly used for community search [1], [20]–[23], [50]. The studies in [1] and [20] further enhance the cohesiveness of $k$-truss by triangle connectivity. They show that the $k$-truss augmented with the triangle-connectivity constraint is closer to practical case and more consistent with sociological studies. The definition of triangle-connected $k$-truss used in this paper follows [20] and [1], which requires the $k$-truss subgraph to be triangle-connected. Recently, $k$-truss community search is also studied on directed graphs [28]. There are other metrics based on vertex pairwise similarity [9], [10], etc. [26], [40] recently study the truss-based community search problem on bipartite graphs.

**Size-constrained community search.** Community search is studied on different graphs [17], including undirected graphs [13], [36], directed graphs [18], attributed graphs [16], multi-dimension graphs [27], and so on. Due to the limitation of budget or capability in many real applications, an important type of community search is size-constrained (size-bounded) community search [17]. In [36], a heuristic algorithm is designed to repetitively removing vertices from a large initial subgraph. A local search algorithm is proposed in [4] by greedily expanding a subgraph starting from the query vertices. Another local search algorithm finds a $k$-core with size equal to $h$ and the smallest closeness among all size-$h$ subgraphs [30]. It is infeasible to apply their algorithms to our problem because a qualified result (SCkT) may be missed.

To further reduce the size of returned communities, [4] and [13] also study the search of minimum communities. An approximate solution for finding a minimum $k$-core is developed in [24]. The above algorithms do not have a certain threshold on limiting the size of each result, and the techniques are not designed for community search with a size threshold. To our best knowledge, this paper is the first work to study the SCkT search. As shown in our experiments, the communities retrieved by the size-constrained triangle-connected $k$-truss model are more promising than that of other models.

## III. Preliminaries

In this section, we introduce the notations, formally define the problem, and analyze the problem hardness.

### A. Notations and Definitions

Let $G = (V, E)$ be an undirected graph, where $V$ is the set of vertices and $E \subseteq V \times V$ is the set of edges. We use $V(G)$ and $E(G)$ to denote the set of vertices and the set of edges of graph $G$, respectively. For a vertex $v \in V$, we denote the set of adjacent vertices (neighbors) of $v$ in $G$ by $N(v, G) = \{u \mid (u, v) \in E(G)\}$ and its degree in $G$ by $d(v, G) = |N(v, G)|$. A triangle $\triangle(u, v, w)$ in $G$ is a cycle of length 3 such that $\{(u, v), (v, w), (w, u)\} \in E(G)$. We say an edge $e$ is incident to vertex $v$ if $v$ is one of the two endpoints of $e$. We say $G'$ is a subgraph of $G$ induced by vertex set $M$ if $V(G') = M$ and $E(G') = (M \times M) \cap E(G)$.

Table I summarizes the notations used throughout the paper. We may omit the input graph $G$ in notations if the context is clear, e.g., using $N(v)$ instead of $N(v, G)$.

TABLE I
Summary of Notations

| Notation | Definition |
|---|---|
| $G(V, E)$ | an unweighted and undirected graph |
| $n$, $m$ | the number of vertices and edges in $G$, respectively (assume $m > n$) |
| $u$, $v$ | a vertex in the graph |
| $N(v, G)$ | the set of adjacent (neighboring) vertices of $v$ in $G$ |
| $d(v, G)$ | the degree of $v$ in $G$ |
| $e$; $(u, v)$ | an edge in the graph; the edge with $u$ and $v$ as endpoints |
| $sup_G(u, v)$ | number of triangles in $G$ containing $(u, v)$ |
| $\tau(e)$ | the trussness of edge $e$ |
| $gap_d(u)$ | the degree gap of vertex $u$ |
| $gap_s(u)$ | the structure gap of vertex $u$ |
| $G_{h,k}$ | $h$-hop truss neighborhood of query vertex $q$ |
| $f(u)$ | score of $u$ for selection in the search |

**Definition 1.** ***Edge Support.*** *The support of an edge $(u, v)$ in G, denoted by $sup_G(u, v)$, is the number of the triangles in G which contain $(u, v)$, that is $sup_G(u, v) = |\{w \mid w \in V(G) \wedge \triangle(u, v, w) \in G\}|$.*

The triangle-connected $k$-truss community in [1], [20] is defined as follows.

**Definition 2.** ***Triangle Connectivity.*** *We say that two edges $e_1$, $e_2$ are triangle-connected in G if either $e_1$ and $e_2$ belong to the same triangle or there exists a series of triangles $\triangle_1, ..., \triangle_n$ in G such that $e_1 \in \triangle_1$, $e_2 \in \triangle_n$ and for any $1 \leq i < n$, $\triangle_i$ and $\triangle_{i+1}$ share a common edge.*

**Definition 3.** ***Triangle-connected $k$-Truss.*** *Given a graph G and an integer k, a triangle-connected k-truss is a subgraph $H \subseteq G$, such that $\forall e \in E(H)$, $sup_H(e) \geq k - 2$ and $\forall e_1, e_2 \in E(H)$, $e_1$ and $e_2$ are triangle-connected.*

**Definition 4.** ***Maximal Triangle-connected $k$-Truss.*** *Given a graph G and an integer k, a maximal triangle-connected k-truss is a triangle-connected k-truss H, such that $\forall I \subseteq E(G)/E(H)$, the subgraph induced by $E(H) \cup I$ is not a triangle-connected k-truss.*

The definition of triangle-connected $k$-truss implies that the incident vertices of each edge share at least $(k - 2)$ common neighbors in the subgraph, which means a triangle-connected $k$-truss is a subgraph of the ($k$-1)-core. Furthermore, the vertex/edge removal in one maximal triangle-connected $k$-truss does not affect the edge support of edges in other triangle-connected $k$-truss as the edge support is defined on triangle and there is no triangle (formed by edges with support no less than $k - 2$) between two maximal triangle-connected $k$-truss. Hence, we have the following observation.

**Observation 1.** *Suppose there are more than one maximal triangle-connected k-truss in graph G, the removal of any vertex/edge in one maximal triangle-connected k-truss does not lead to removal of vertices/edges in other maximal triangle-connected k-trusses.*

**Definition 5.** ***Trussness.*** *The trussness of an edge $e \in E(G)$, denoted by $\tau(e)$, is the largest k such that a triangle-connected*

*$k$-truss contains $e$.*

**Definition 6.** ***Size-constrained triangle-connected $k$-truss community (`SCkT`).*** *Given a graph $G$, a set of query vertices $Q \subset V$, a trussness constraint $k$, and a size constraint $s$, $H$ is a size-constrained triangle-connected $k$-truss community (`SCkT`) if (1) $H$ is a triangle-connected $k$-truss containing $Q$, i.e., $Q \subseteq V(H) \subseteq V(G)$; (2) the number of vertices in $H$ is no larger than $s$, i.e., $|V(H)| \leq s$.*

A direct observation of `SCkT` is that if a `SCkT` exists, $s$ must be no less than $k$ since the degree of each vertex in a triangle-connected $k$-truss is no less than $k-1$.

**Problem Statement.** Given a graph $G$, a set of query vertices $Q \subseteq V(G)$, a trussness constraint $k$, and a size constraint $s$, the `SCkT` search problem aims to find a `SCkT` from $G$, i.e., a triangle-connected $k$-truss subgraph $H$ of $G$ s.t. $H$ contains $Q$ and the number of vertices in $H$ does not exceed $s$.[1]

### B. Problem Hardness

We prove the NP-hardness of `SCkT` search problem.

**Theorem 1.** *The `SCkT` search problem is NP-hard.*

*Proof.* We prove this by reducing the decision version of maximum clique problem to the decision version of `SCkT` search problem. Given a graph $G$ and an integer $k$, the maximum clique decision problem is to check whether $G$ contains a clique of size $k$. We construct an instance of `SCkT` search problem consisting of $Q = \emptyset$, parameters $k$, and $s = k$. Clearly, any clique with $k$ vertices is a triangle-connected $k$-truss with size $k$. On the other hand, any solution to the instance of `SCkT` search problem is a clique of size $k$ because the number of vertices in the solution must equal to $k$. Therefore, the instance of the maximum clique decision problem is a YES-instance iff the corresponding instance of `SCkT` search problem is a YES-instance. □

For presentation simplicity and the ease of understanding, in the reminder of the paper, we focus on the `SCkT` search containing one query vertex $q$, and then discuss the search of `SCkT` for multiple query vertices in Section IV-E.

## IV. Our Solution

Despite the NP-hardness of the `SCkT` search problem, as the first study of the `SCkT` search, we explore the possibility of an efficient exact solution. In this section, we present an exact solution through exploiting the nature of triangle-connected $k$-truss on real-life data. The solution can be efficient in practice for almost all the queries. The framework of our solution is introduced in Section IV-A. To prune unpromising search branches, a novel lower bound of triangle-connected $k$-truss size is developed in Section IV-B. According to different situations of current partial solution, two search strategies are designed including the expansion-based search (Section IV-C) and the shrinking-based search (Section IV-D). A hybrid search method combining the expansion and shrinking strategies is proposed in Section IV-E.

**Algorithm 1**: **Framework**

```
Input  : a graph $G$, the trussness of each edge $e \in E(G)$, the
         trussness constraint $k$, the size constraint $s$, the query
         vertex $q$
Output : SCkT $R$
$h \leftarrow 1$;
while $G_{h,k} \supsetneq G_{h-1,k}$ AND $h \leq \lfloor \frac{2s-2}{k} \rfloor$ do
    for each maximal triangle-connected $k$-truss $G'$ of $G_{h,k}$
    containing $q$ do
        $M \leftarrow \{q\}$;
        $R \leftarrow$Search($G', M, k, s$);
        if $R \neq \emptyset$ then
            return $R$
    $h \leftarrow h + 1$;
return $\emptyset$
```

### A. The Framework

Starting from the query vertex $q$, we process the graph progressively according to the number of hops from the query vertex. Namely, we first search `SCkT` from the neighbors of $q$ (i.e., 1-hop), if no `SCkT` can be found we enlarge the search space to the 2-hop neighbors of $q$. This is conducted until a result is returned, or we can safely claim there does not exist a qualified `SCkT` containing $q$.

The rationale for adopting this framework is twofold. Firstly, triangle-connected $k$-truss is cohesive and the vertices in the resulting `SCkT` are usually very close to the query vertex. By restricting the distance between the vertices and the query vertex $q$, the framework can largely reduce the search space. Secondly, the radius of a triangle-connected $k$-truss subgraph is bounded regarding trussness constraint $k$ and size constraint $s$, s.t., we can safely terminate the algorithm when $h > \lfloor \frac{2s-2}{k} \rfloor$ because there is certainly no `SCkT` in such $h$-hop truss neighbors of $q$. In the following, we formally introduce this property of `SCkT`.

**Definition 7.** ***$h$-Hop Truss Neighborhood.*** *Given a graph $G$, a query vertex $q$ and the $k$-truss $S$ containing $q$, the $h$-hop truss neighborhood of $q$, denoted as $G_{h,k}$, is the subgraph induced by the vertices from $S$ within distance $h$ to $q$, i.e., $V(G_{h,k})$ = $\{v \in V(S) \mid dist(v, q, S) \leq h\}$ where $dist(v, q, S)$ is the length of a shortest path from $v$ to $q$ in $S$.*

Then, we have the following lemma for a `SCkT` in $G_{h,k}$.

**Lemma 1.** *If there does not exist a `SCkT` in the $h$-hop truss neighborhood $G_{h,k}$ of query vertex $q$ for any $h \leq \lfloor \frac{2s-2}{k} \rfloor$, there does not exist a `SCkT` containing $q$ in the whole graph $G$ and the search can be safely terminated.*

*Proof.* The lemma is immediate because the diameter of a `SCkT` $S$ in $G$ is at most $\lfloor \frac{2s-2}{k} \rfloor$ [11], [22], and $S$ is always a subgraph of $G_{h,k}$ for a query vertex $q$. □

The pseudo-code of the framework is given in Algorithm 1. The trussness of each edge in $G$ is pre-computed by truss decomposition [11], which is executed only once to support different queries. Then, we can fast retrieve a $h$-hop truss

[1] Setting a suitable size constraint is indeed an important question. In some cases when the size constraint is related to user budget, we may consider to set the constraint by user-specified values. It is also interesting to determine the threshold automatically, if there is a goodness metric in specific applications.

neighborhood $G_{h,k}$ by searching the neighbors of $q$ through edges with trussness of at least $k$. If the whole graph has been searched or $h$ becomes larger than the diameter bound, the search terminates (line 2). In each of the $G_{h,k}$ subgraph starting from $G_{1,k}$ (i.e., subgraph formed by $q$ and its 1-hop neighbors in triangle-connected $k$-truss), we process each maximal triangle-connected $k$-truss separately since they are independent with each other as shown in Observation 1 (line 3). We use $M$ to record all the vertices in the partial solution of SCkT and $M$ is initialized as $\{q\}$ (Line 4). Then, we call the Search algorithm (introduced in Section IV-C, IV-D and IV-E) to explore a SCkT on $G'$ (Line 5). If a non-empry set $R$ is returned by the search on $G'$, $R$ is a SCkT and the algorithm is returned (line 6-7).

### B. Lower Bound Computation

In this subsection, we introduce a novel and effective lower bound for a partial solution $M$ of SCkT, which aims to calculate the smallest possible size of a SCkT containing all vertices in $M$. This lower bound can thus be used to early terminate unpromising search branches to improve efficiency. Regarding each vertex $v \in M$, a key issue to develop a tight bound is computing the number of extra vertices that we need to include outside $M$ such that $v$ satisfies triangle-connected $k$-truss constraint.

In [24], Li et al. propose a lower bound predication algorithm based on degree constraint, which is called IELB. Suppose that the degree constraint on the final solution is that each vertex must have degree at least $d$ (for triangle-connected $k$-truss $d = k - 1$). The degree gap for each vertex $v$ in $M$ is defined as follows:

**Degree Gap.** Let $G$ denote the target graph to compute a solution from it, $M$ denote the vertices in the partial solution and $G_M$ denote the subgraph induced by $M$ on $G$. Suppose that the degree constraint on the final solution is that each vertex must have degree at least $d$. For each vertex $v \in M$, the degree gap of $v$, $gap_d(v)$, is defined as:

$$gap_d(v) = \max\{d - deg(v, G_M), 0\} \tag{1}$$

Let $X$ be the set of vertices outside of $M$ but incident to $v$, e.g., $X = N(v, G)/M$. IELB first computes $gap_d(v)$ for each $v \in M$, then it calculates the lower bound using an inclusion-exclusion method. Specifically, the lower bound $l$ is initialized as $|M|$. IELB selects the vertex $v$ from $M$ with the largest degree gap $gap_d(v)$, and updates $l = l + gap_d(v)$. For other vertices $u \in M$, IELB updates $gap_d(u) = \max\{gap_d(u) - |N(u, G) \cap X|, 0\}$. The main idea behind this is that IELB assumes that any vertex from $X$ is incident to other vertices in $M$ as many as possible. IELB keeps updating $l$ until the degree gap for each vertex in $M$ equals to 0 and uses $l$ as the lower bound.

**Discussion.** As shown in [24], IELB is more efficient than other algorithms in predicting lower bound on $k$-core model. However, IELB suffers from two drawbacks when applied to triangle-connected $k$-truss model. 1) the degree gap is not tight enough since triangle-connected $k$-truss is defined based on triangle rather than degree. 2) The connection between vertices in triangle-connected $k$-truss is closer than that in $k$-core, and the effectiveness of inclusion-exclusion method used in IELB is reduced. This is because after selecting the vertex $v$ with the largest degree gap, vertices from $X = N(v, G)/M$ are incident to other vertices in $M$ with high probability which often makes $\max\{gap_d(u) - |N(u, G) \cap X|, 0\}$ equal to 0. Therefore, under most cases, IELB degenerates to using $|M| + max\{gap_d(v)|v \in M\}$ as lower bound.

Considering the above drawbacks, we introduce the structural gap by exploring the structural properties of triangle-connected $k$-truss to get a tighter bound. Recall that the edge support of an edge $(u, v)$ is the number of triangles containing $(u, v)$. To utilize the edge supports in lower bound computation, we first introduce the locality property of the triangle-connected $k$-truss [20].

**Property 1.** *For any vertex $u$ in a triangle-connected $k$-truss $S$, there exist at least $k-1$ neighbors of $u$ such that the support of the edges between them and $u$ is no less than $k - 2$, i.e., $|\{v \in N(u) \mid sup_S(u, v) \geq k - 2\}| \geq k - 1$.*

Property 1 allows us to determine whether a vertex $u$ exists in a triangle-connected $k$-truss by locally looking at the subgraph induced by $u$ and $N(u)$. The following lemma introduces a lower bound based on the locality property.

**Lemma 2.** *Given a graph $G$, if a vertex $u$ is contained in a triangle-connected $k$-truss, let $G_u$ denote the subgraph induced by $\{u\} \cup N(u)$ on $G$, after iteratively removing every edge $(u, v)$ incident to $u$ satisfying $sup_{G_u}(u, v) < k - 2$, $u$ must have at least $k - 1$ remaining neighbors.*

*Proof.* If $sup_{G_u}(u, v) < k - 2$, $(u, v)$ cannot be included in any triangle-connected $k$-truss. So, we can iteratively remove every edge $(u, v)$ from $G_u$ with $sup_{G_u}(u, v) < k - 2$. By Property 1, $u$ must have at least $k - 1$ remaining neighbors after the removals. □

We are now ready to introduce the structure gap for each vertex $u$ in the partial solution $M$. Let $G'$ denote the target graph to compute a SCkT from it. For $u \in M$, we use $G'_u$ to denote the subgraph of $G'$ induced by $\{u\} \cup N(u, G') \cap M$.

**Structure Gap.** The structure gap of a vertex $u$, denoted as $gap_s(u)$, is the least number of vertices to be added into $G'_u$ in order to make $u$ contained in a triangle-connected $k$-truss based on Lemma 2. For each vertex $x$ to be added, we make following two assumptions: (1) $x$ is connected with all the vertices in $V(G'_u)$, i.e., the vertices in $\{u\} \cup N(u, G') \cap M$; (2) The edge support of $(u, x)$ is always large enough, i.e., $sup(u, x) = \infty$. We call such vertex $x$ an *ideal vertex* and use $idv$ to denote the number of ideal vertices added into $M$.

Why ideal vertices. We apply the ideal vertices to compute the lower bound due to following reasons: (1) the triangle-connected $k$-truss is usually dense, thus we assume that $x$ is connected with all the vertices in $G'_u$ for computation efficiency; (2) every vertex $x$ added into $M$ is included in the final solution, hence we assume $x$ is always a neighbor of $u$ in triangle-connected $k$-truss, i.e., $sup(u, x) = \infty$.

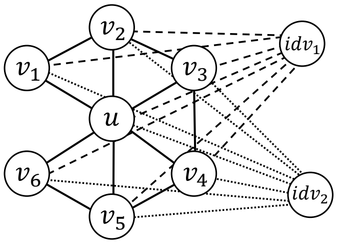


Fig. 2. Lower Bound Example of $u$ with $k = 5$

Structure gap computation. To compute $gap_s(u)$ for a vertex $u \in M$, we increase $idv$ one by one until $u$ meets the condition in Lemma 2 and set $gap_s(u)$ as $idv$. Specifically, for each vertex $u \in M$, we find the minimum value of $idv$ such that $u$ still has at least $k-1-idv$ neighbors after iteratively removing every edge $(u, v)$ in $G'_u$ subject to $sup(u, v) + idv < k - 2$. We illustrate this with the following example:

**Example 1.** *We give an example of $G'_u$ in Figure 2, which consists of $v_1$, $v_2$, $v_3$, $v_4$, $v_5$, $v_6$, and u. Note that since the degree of u is 6 larger than $k-1 = 4$, $gap_d(u) = 0$. The supports of edges incident to u are $sup(u, v_1) = sup(u, v_6) = 1$ and $sup(u, v_2) = sup(u, v_3) = sup(u, v_4) = sup(u, v_5) = 2$. Since $k = 5$ and the number of edges incident to u with support no less than $k - 2 = 3$ is 0, we add one ideal vertex $idv_1$ to $G'_u$. Now, the support of each edge is increased by 1, and we have $sup(u, v_2) = sup(u, v_3) = sup(u, v_4) = sup(u, v_5) = 3$. As the newly added ideal vertex is also a neighbor of u, there are 5 neighbors of u satisfying the support of edges between them and u is no less than 3. At this point, Property 1 is satisfied. However, according to Lemma 2, after removing edges $(u, v_1)$ and $(u, v_6)$ whose support is less than 3, the supports of $(u, v_2)$ and $(u, v_5)$ also become less than 3. Finally, we only have one edge $(u, idv_1)$ left as we assume $sup(u, idv_1) = \infty$. Hence, we add another ideal vertex $idv_2$. This time, all the edges $(u, \cdot)$ have support no less than 3 and Lemma 2 is satisfied. Therefore, $gap_s(u)$ is 2.*

**Lemma 3.** $\forall u \in M, gap_s(u) \geq gap_d(u)$.

*Proof.* When $gap_d(u) = k - 1$ or 0, it is obvious that $gap_s(u) = gap_d(u)$. Suppose that $0 < gap_d(u) = t < k - 1$, $u$ has $k - 1 - t$ neighbors in $G'_u$. Therefore, the largest support of each edge incident to $u$ in $G'_u$ is $k - 2 - t$ when all the neighbors of $u$ form a clique. According to the computation process of structure gap, we need at least $k - 2 - (k - 2 - t) = t$ ideal vertices to make $u$ satisfy Lemma 2. Thus, $gap_s(u) \geq t = gap_d(u)$. In summary, we have $\forall u \in M, gap_s(u) \geq gap_d(u)$. □

The details for lower bound computation regarding partial solution $M$ is shown in Algorithm 2. For each vertex $u$ in the partial solution $M$, its degree gap is calculated by visiting the neighbor set of $u$ to count $deg(u, G'_M)$ where $G'_M$ is subgraph of $G'$ induced by $M$. If $|M| + \max\{gap_d(u)|u \in M\}$ exceeds the size constraint $s$, there is no `SCkT` in current search space ($G'$). If so, the degree gap based lower bound is returned and the computation is completed (Line 1-2).

**Algorithm 2**: **LowerBound**

```
Input  : a graph $G'$, a partial solution of SCkT $M$, the
         trussness constraint $k$, the size constraint $s$
Output : the lower bound of the size of every
         triangle-connected $k$-truss containing $M$ in $G'$
1  if $|M| + \max\{gap_d(u)|u \in M\} > s$ then
2      return $|M| + \max\{gap_d(u)|u \in M\}$
3  for each $u \in M$ do
4      $G'_u \leftarrow$ the subgraph of $G'$ induced by $u \cup N(u, G') \cap M$;
5      $idv \leftarrow \min\ t$ s.t. $|\{sup_{G'_u}(u, v) + t \geq k - 2\}| + t \geq k - 1$;
6      while True do
7          if $|M| + idv > s$ then
8              return $|M| + idv$
9          $H \leftarrow G'_u$; iteratively remove every edge $e$ incident to
           $u$ from $H$ satisfying $sup_H(e) + idv < k - 2$;
10         if $|N(u, H)| + idv \geq k - 1$ then
11             $gap_s(u) \leftarrow idv$;
12             break;
13         $idv \leftarrow idv + 1$;
14 return $|M| + \max\{gap_s(u)|u \in M\}$
```

For each vertex $u \in M$, its structure gap is calculated according to the locality property and Lemma 2 (Line 3-13). For a vertex $u$, we first retrieve the subgraph $G'_u$ induced by $u$ and its neighbors in $M$ (Line 4). The initial value of ideal vertices ($idv$) is calculated following Property 1 (Line 5). Once the lower bound is larger than $s$, the computation is immediately returned (Line 7-8).

Line 9-13 applies Lemma 2 to compute the structure gap for $u$. It iteratively removes every edge $e$ incident to $u$ in $H$ satisfying $sup_H(e) + idv < k - 2$, because such edges cannot exist in any triangle-connected $k$-truss of $G'$ containing $u$ (Line 9). If $|N(u, H)| + idv \geq k - 1$, we know that $idv$ equals to $u$'s structure gap and the computation is finished (Line 10-12). Otherwise, we lift $idv$ by one (Line 13) and repeat the process.

After the structure gap for every vertex $u \in M$ is computed, the maximum structure gap plus the size of $M$ is returned as the lower bound of the size of every triangle-connected $k$-truss containing $M$ in $G'$ (Line 14).

We prove the correctness of Algorithm 2 in the following.

**Theorem 2.** *The value returned by Algorithm 2 is a lower bound of the size of every triangle-connected k-truss S with $M \subset S$ and $S \subseteq G'$.*

*Proof.* (1) If Algorithm 2 returns at Line 2, let $b = \max\{gap_d(u)|u \in M\}$, adding $b$ vertices to $M$ is possible to make every vertex in $M$ to stay in a triangle-connected $k$-truss of $G'$. Thus, $|M| + \max\{gap_d(u)|u \in M\}$ is a correct lower bound. (2) If Algorithm 2 returns at Line 8, we need at least $idv$ vertices to be added in $M$ s.t. $u$ can exist in a triangle-connected $k$-truss of $G'$. Thus, $|M| + idv$ is a correct lower bound. (3) If Algorithm 2 returns at Line 14 and $u$ is the vertex with the largest structure gap, according to Lemma 2, $gap_s(u)$ is the smallest number of ideal vertices to be added in $M$ s.t. $u$ can stay in a triangle-connected $k$-truss of $G'$. Hence, $|M| + \max\{gap_s(u)|u \in M\}$ is a lower bound. □

**Complexity.** The dominating cost of Algorithm 2 is the

**Algorithm 3: Expand**

```
Input  : a graph $G'$, the set $M$ of vertices that must be
         included, $k$, $s$
Output : the triangle-connected $k$-truss $R$ containing $M$ with
         size no larger than $s$ if exists, otherwise $\emptyset$
1  $lb \leftarrow$ LowerBound($M, G', k, s$);
2  if $lb > s$ then
3      return $\emptyset$
4  else if $\forall v \in M, gap_s(v) = 0$ then
5      $H \leftarrow$ the subgraph of $G'$ induced by $M$;
6      $R \leftarrow$ remove edges $e$ with $sup_H(e) < k - 2$ from $H$;
7      if $V(R) = M \wedge R$ is triangle-connected then
8          return $R$
9  while True do
10     $u \leftarrow$ the vertex in $V(G') \setminus M$ with the largest score
       (Definition 8);
11     $R \leftarrow$ Expand($G', M \cup \{u\}, k, s$);
12     if $R \neq \emptyset$ then
13         return $R$
14     $G' \leftarrow$ RemoveVertex($G', M, u, k$);
15     if $V(G') \cap M \neq M$ then
16         return $\emptyset$
```

**Algorithm 4: RemoveVertex**

```
Input  : a graph $G'$, the vertex $u$ to be removed
Output : updated $G'$ and $\{sup_{G'}(\cdot)\}$
1  $S \leftarrow \emptyset$;
2  for each $v \in N(u, G')$ do
3      $S.push((u, v))$;
4  while $S \neq \emptyset$ do
5      $(x, y) \leftarrow S.pop()$;
6      remove $(x, y)$ from $G'$;
7      for $w \in N(x, G') \cap N(y, G')$ do
8          $sup_{G'}(x, w) \leftarrow sup_{G'}(x, w) - 1$;
9          if $sup_{G'}(x, w) < k - 2 \wedge (x, w) \notin S$ then
10             $S.push(x, w)$;
11         Line 8-10 by exchanging $x$ with $y$;
12 remove isolated vertices from $G'$;
13 return $G'$
```

computation of $gap_s(u)$ in Line 6-13. At Line 10, the largest possible value of $idv$ is $k-1$ since $idv$ is lifted one by one. Therefore, Line 6-13 can be executed at most $k-1$ times. The time cost of a single iteration is $O(s^2)$ because $|V(H)| \leq |M| \leq s$ and all the edges incident to $u$ can be iteratively removed in $O(|E(H)|)$, which is no larger than $|V(H)|^2 \leq s^2$. Considering that there are no more than $s$ vertices in $M$, Line 3 can be executed at most $s$ times and the time cost of Line 3-13 is $O(ks^3)$. In summary, the time cost of Algorithm 2 is $O(ks^3)$ where $k$ and $s$ are constant values.

### C. Expansion-based Search Algorithm

The intuition of the expansion-based search is that starting from the query vertex $q$, we iteratively expand with the most promising vertex to form a SCkT. A scoring function is designed to measure the "goodness" that a vertex will bring to the current partial solution. The search follows a back-track manner to effectively explore the search space and avoid duplicate computation.

In each iteration, we select the most promising vertex from $V(G') \setminus M$ and add it into $M$ until $M$ forms a triangle-connected $k$-truss. If we find that the addition of a vertex makes the lower bound of $M$ larger than the size constraint $s$, we remove this vertex from $G'$ and $M$. Note that during the search process, we always keep $G'$ as a triangle-connected $k$-truss. Thus, the removal of a single vertex may lead to the removal of other vertices.

**Vertex Selection.** We consider two criteria when selecting the vertex. The first criterion is how much contribution the vertex will make to help $M$ form a triangle-connected $k$-truss. For the first criterion, we count the number of edges between the selected vertex and the vertices whose structure gap is larger than 0. The second criterion is how many additional vertices needed to make the selected vertex included in a triangle-connected $k$-truss. For the second criterion, we count the number of edges between the selected vertex and the vertices in $M$. Intuitively, vertex with more edges requires less additional vertices to make it exist in a triangle-connected $k$-truss. Consequently, we define the following score function to determine which vertex will be selected into $M$.

**Definition 8.** ***Score Function.*** *For every vertex $u \in V(G') \setminus M$, let $N_u$ denote $N(u, G') \cap M$, the goodness of selecting $u$ into $M$ is defined as:*

$$f(u) = |\{v | v \in N_u \wedge gap_s(v) > 0\}| + |N_u| \tag{2}$$

第一项：可以填补已有空缺，第二项：该点不需要额外开销

Algorithm 3 shows the pseudo-code of expansion-based search. It first computes the lower bound $lb$ for current partial solution $M$ by invoking LowerBound (Line 1). If the lower bound is larger than $s$, it returns an empty set as there is no feasible SCkT containing $M$ with size not larger than $s$ (line 2-3). Otherwise, if the structure gap for each vertex in $M$ is 0, it checks whether the vertices in $M$ can form a triangle-connected $k$-truss (Line 4-8). If not, Algorithm 3 selects the vertex $u$ with the largest score (Line 10) and recursively invokes Expand to compute a SCkT containing $M \cup \{u\}$ with size no larger than $s$ (Line 11). If a SCkT is found, Algorithm 3 simply returns it as the result (Line 12-13). Otherwise, $u$ is removed from $G'$ because $u$ cannot be included in any feasible result. Other vertices may be removed together with $u$ to make sure $G'$ is a triangle-connected $k$-truss (Line 14). If the removal of $u$ leads to the removal of any vertex in $M$, Algorithm 3 returns an empty set (Line 15-16).

The pseudo-code of RemoveVertex invoked at Line 14 is shown in Algorithm 4. It first pushes all the edges incident to $u$ into a stack $S$ (Line 1-3). Then for each edge $(x, y)$ in $S$, Algorithm 4 removes it from $G'$ and decreases the edge support by 1 for those edges which form a triangle with $(x, y)$ (Line 4-8). If the edge support of an edge is less than $k-2$, Algorithm 4 pushes it into $S$ (Line 9-10). After removing all the edges which are no longer contained in the triangle-connected $k$-truss, it removes isolated vertices from $G'$ (Line 12). Finally, Algorithm 4 returns the updated $G'$ (Line 13). Note that the support of each edge in $G'$ can be dynamically maintained during the search process and we do not need to recount it in Algorithm 4.

**Remark.** The while loop in Algorithm 3 always terminates as either a SCkT will be found or a vertex in $M$ will be deleted. If a SCkT $R$ is found by Line 11, the algorithm returns at Line 13. If there is no SCkT in $G'$, every vertex in $V(G') \setminus M$ will be deleted at Line 14. Since $M$ alone cannot form a triangle-connected $k$-truss which has been verified in Line 4-8, after all the vertices in $V(G') \setminus M$ are removed, some vertices in $M$ must be removed due to the triangle-connected $k$-truss constraint. At this point, the algorithm returns at Line 16.

**Correctness.** Suppose that there is a SCkT $R$ containing $M$. If $V(R) = M$, Algorithm 3 will return $R$ at Line 8. Otherwise, unless there exists other solutions, whenever a vertex not in $V(R) \setminus M$ is added to $M$, Algorithm 3 will remove it at Line 14. Thus, it always holds that $R \subseteq G'$ because only those vertices and edges not contained in $R$ are removed at Line 14. Finally, either all the vertices $V(R) \setminus M$ are added to $M$ at Line 11 or all the vertices outside $V(R)$ are removed at Line 14. So, Algorithm 3 always finds the solution $R$ containing $M$.

**Complexity.** The number of triangles in $H$ can be expressed as $\rho \times |E(H)|$ where $\rho$ is the arboricity [20] of $H$. We can test whether $M$ is a solution or not at Line 4-8 in $O(\rho \times |E(H)|)$. Since there are at most $s^2$ edges in $H$ and the arboricity $\rho \leq \sqrt{|E(H)|} \leq s$, the time complexity of Line 4-8 is bounded by $O(s^3)$. As there are at most $\rho \times |E(G')|$ triangles, the time cost of Line 14 is $O(|E(G')|^{1.5})$. Let $T_i$ denote the time complexity of Algorithm 3 when $|M| = i$. Considering that LowerBound runs in $O(ks^3)$ and Line 9-16 can be executed at most $|V(G')|$ times, we have $T_i = |V(G')| \times T_{i+1} + O(|E(G')|^{1.5} + ks^3)$. Since $i$ cannot be larger than $s$, we have $T_{|M|} = O(|V(G')|^{s-|M|} \times (|E(G')|^{1.5} + ks^3))$. At Line 5 in Algorithm 1, we have $|M| = 1$, $|V(G')| = O(n)$, and $|E(G')| = O(m)$, thus the time cost for querying one single vertex with Algorithm 3 is $O(n^{s-1} \times (m^{1.5} + ks^3))$.

### D. Shrinking-based Search Algorithm

When the size constraint $s$ and trussness constraint $k$ become relatively large, the expansion-based search algorithm may incur higher cost, because it keeps adding vertices until a SCkT is found. In such cases, a shrinking-based search algorithm may be preferred due to the small search space. The shrinking algorithm iteratively removes a single vertex which leads to the cascading removal of other vertices due to the triangle-connected $k$-truss constraint.

To adopt the shrinking-based search strategy, we replace Line 5 of Algorithm 1 with Shrink. The pseudo-code of Shrink is given in Algorithm 5. It first selects the vertex $u$ with the smallest score as defined in Definition 8 as smaller score means looser connection to current partial solution (line 2). If $u$ can be removed from $G'$ (Line 3-4), it checks whether the size of remaining non-empty graph is no larger than $s$. If the answer is yes, it returns current $G'$ as the solution (Line 5-7). Otherwise, Algorithm 5 recursively invokes itself to find a solution based on the remaining graph (Line 8-10). If $u$ cannot be removed from $G'$ at Line 3 or no solution can be found at Line 8, i.e., $R = \emptyset$, Algorithm 5 inserts $u$ and all the vertices removed together with it back into $G$ and adds $u$ to $M$ (Line 11-12). Then it computes the lower bound of current $M$ (line 13). If the lower bound of $M$ on $G'$ is larger than $s$, current search branch is terminated and returned (Line 14-15).

**Algorithm 5**: Shrink

```
Input  : a graph $G'$, the set $M$ of vertices that must be
         included, $k$, $s$
Output : the triangle-connected $k$-truss $R$ containing $M$ with
         size no larger than $s$ if exists, otherwise $\emptyset$
1  while True do
2      $u \leftarrow$ the vertex with the smallest score in Definition 8;
3      $G' \leftarrow$ RemoveVertex($G', M, u, k$);
4      if $M \subseteq V(G')$ then
5          if $V(G') \leq s \wedge G'$ is triangle-connected then
6              $R \leftarrow G'$;
7              return $R$
8          $R \leftarrow$ Shrink($G', M, k, s$);
9          if $R \neq \emptyset$ then
10             return $R$
11     insert $u$ and vertices removed with $u$ back into $G'$;
12     $M \leftarrow M \cup \{u\}$;
13     $lb \leftarrow$ Lowerbound($M, G', k, s$);
14     if $lb > s$ then
15         return $\emptyset$
```

**Algorithm 6**: Hybrid

```
Input  : a graph $G'$, the set $M$ of vertices that must be
         included, $k$, $s$
Output : the triangle-connected $k$-truss $R$ containing $M$ with
         size no larger than $s$ if exists, otherwise $\emptyset$
1  if $|\{u \mid u \in V(G') \setminus M \wedge f(u) \geq |M|\}| > \frac{1}{2}|V(G') \setminus M|$ then
2      Alg. 3 where Expand at line 11 is replaced with Hybrid;
                                  至少有一半节点是promising的
3  else
4      Alg. 5 where Shrink at line 8 is replaced with Hybrid;
```

**Correctness.** Suppose that there is a solution $R$ containing $M$. Removing vertices in $V(G') \setminus V(R)$ will not lead to removal of any vertex in $M$ by triangle-connected $k$-truss constraint. Hence, before Algorithm 3 selects any vertex in $R$ at Line 2, it continues invoking itself at Line 8 to compute a solution based on $G'$ which contains $R$. Once a vertex $u$ in $V(R)$ is selected at Line 2, unless there exist other solutions, the returned result at Line 8 must be empty and $u$ will be added to $M$ at Line 12. Therefore, Algorithm 3 can always find a solution.

**Complexity.** There are at most $s$ iterations for Algorithm 3 as $|M|$ cannot be larger than $s$ and $|M|$ is increased by 1 by the end of each iteration. Thus, the time cost except Line 8 is $O(ks^4 + |E(G')|^{1.5})$. Let $T_i$ be the time cost of Algorithm 3 when $|V(G')| = i$, we have $T_i = s \times T_{i-1} + O(ks^4 + |E(G')|^{1.5})$. Since $i$ is no smaller than $s$, the time complexity of querying a single vertex with Algorithm 3 is $O(s^{n-s}(m^{1.5} + ks^4))$. Although the time cost of Algorithm 5 is higher than Algorithm 3, we found that the performance of Algorithm 5 is better in many settings because removing the vertices inside a triangle-connected $k$-truss may quickly lead to its collapse, and thus the search space is reduced.

### E. The Hybrid Approach

Although Shrink is efficient in the sense that removing some vertices can lead to the removal of other unpromising vertices,

TABLE II
STATISTICS OF DATASETS

| Dataset | Type | $\lvert V \rvert$ | $E$ | $d_{avg}$ | $d_{max}$ | $k_{max}$ |
|---|---|---|---|---|---|---|
| Epinion | Social | 76K | 405K | 10.6 | 3.0K | 22 |
| Gowalla | Social | 99K | 817K | 9.2 | 10.0K | 23 |
| DBLP | Authorship | 64K | 457K | 25.6 | 1.1K | 36 |
| YouTube | Social | 1.1M | 3.0M | 5.3 | 28.8K | 19 |
| Flickr | Social | 1.7M | 15.6M | 18.2 | 27.2K | 200 |
| Orkut | Social | 3.1M | 117M | 76.3 | 33.3K | 78 |
| Wiki | Hyperlink | 12.1M | 378M | 62.2 | 963K | 1112 |
| Twitter | Social | 41M | 1.5B | 70.5 | 3.1M | 1998 |
| ERGraph | Synthetic | 1M | 100M | 200 | 312 | 43 |
| RMGraph | Synthetic | 40M | 1.0B | 50 | 2.2M | 2437 |

which largely reduces the search space. Expand can quickly find a SCkT if the SCkT is closely connected to current partial solution. Therefore, it is desirable to switch between Shrink and Expand based on specific situations. To this end, we propose a hybrid search strategy which automatically determines the search order of Shrink or Expand based on current partial solution $M$ and subgraph $G'$. According to Definition 8, the score of vertices in $V(G')\backslash M$ ranges from 0 to $2|M|$. If a vertex has score no less than $|M|$, it either connects to all the vertices in $M$ or makes contribution to help $M$ form a triangle-connected $k$-truss. Hence, we recognize that vertices with score no less than $|M|$ are closely connected to partial solution $M$. We choose Expand if the number of such vertices exceeds half of the total number ($\frac{1}{2}|V(G')\backslash M|$) and choose Shrink vice versa. The pseudo-code is given in Algorithm 6. Note that in line 2 and line 4, we replace the "Expand" in Algorithm 3 and "Shrink" in Algorithm 5 with "Hybrid". Therefore Hybrid algorithm is recursively called during the computation to determine the next search branch.

**Multi-vertex Query.** Algorithm 3, Algorithm 5, Algorithm 6 and can be easily extended to support multi-vertex queries. We assume that all the query vertices are triangle-connected with each other, otherwise the query result is empty. At line 4 in Algorithm 1, we put all the query vertices into $M$ and invoke Expand, Shrink, or Hybrid at line 5. The only difference is that instead of using $h$-hop truss neighborhood for single vertex query, we can simply use the union of $h$-hop truss neighborhood of every query vertex or the entire maximal triangle-connected $k$-truss containing query vertices, as the input graph $G'$ for Expand, Shrink, and Hybrid.

**Discussion.** Our proposed algorithms can be easily extended to find any number of SCkTs. In algorithm 3 (Extend) and algorithm 5 (Shrink), we can use a set to store the SCkTs and continue computing other solutions. Specifically, we use a set to collect the SCkT at line 8 of algorithm 3 and line 7 of algorithm 5, and change the "if" condition at line 12 of algorithm 3 and line 9 of algorithm 5 to test whether the number of current SCkTs exceeds the threshold. Besides, our computing framework can be used for future study to find diversified SCkTs (with small overlaps), if candidate SCkTs are updated to be more diverse during the search procedure.

## V. EXPERIMENTAL EVALUATION

In this section, we conduct extensive performance studies to evaluate the effectiveness and efficiency of our algorithms.

### A. Experimental Setting

**Datasets.** We use 8 real-life graphs and 2 synthetic graphs in our experiments. DBLP is downloaded from http://snap.stanford.edu/. Other real-life datasets are from http://konect.uni-koblenz.de/. The synthetic graph ERGraph and RMGraph are generated by the open-sourced graph generator *GTgraph* [2], based on the Erdős-Rényi model [15] and Recursive Matrix model [7], respectively. Table II shows the statistics of the datasets which are ordered by the number of edges. $d_{avg}$ and $d_{max}$ denote the average and maximum degree, respectively. The last column $k_{max}$ is the largest trussness value in the corresponding graph.

**Algorithms.** To the best of our knowledge, no existing work investigates the SCkT search problem and corresponding algorithms. We mainly evaluate three SCkT search algorithms: Expand, Shrink, and Hybrid. In terms of community size, we compare our algorithms with four heuristic algorithms, e.g., RD, DG, VL, and SF. Each heuristic algorithm uses a different vertex selection method. That is, starting from the query vertex, each heuristic algorithm keeps adding vertices based on its vertex selection criterion until it finds a triangle-connected $k$-truss containing the query vertex.

In the experiments, we also test the effectiveness of two lower bounds which are computed based on the IELB [24] and structure gap. Note that if an algorithm uses the lower bound based on the IELB, it simply replaces Algorithm 2 with IELB during the search process.

- Expand: our proposed expansion-based algorithm, e.g., Algorithm 1 + Algorithm 3.
- Shrink: our proposed shrinking-based algorithm, e.g., Algorithm 1 + Algorithm 5.
- Hybrid: our proposed hybrid algorithm, e.g., Algorithm 1 + Algorithm 6.
- Expand-D, Shrink-D, and Hybrid-D: our proposed expansion-based, shrinking-based, and hybrid algorithms which compute lower bound with IELB.
- RD: the heuristic algorithm which randomly selects the vertex to be added.
- DG: the heuristic algorithm which sorts all the vertices according to their degree and selects the vertex with the highest rank.
- VL: the heuristic algorithm which sorts all the vertices by their rank in the vertex layers and selects the vertex with the highest rank. The vertex layers is the removing order of vertices during the $k$-truss decomposition process [48]. A vertex removed from the graph later will have higher rank in the vertex layers. Intuitively, vertices with higher rank form a triangle-connected $k$-truss more easily.
- SF: the heuristic algorithm which selects the vertex with the highest score as defined in Definition 8.

**Parameters.** We conducted experiments under different settings by varying the size constraint $s$ and the trussness constraint $k$. The default values are $s = 30$ and $k = 10$. All the query vertices are from 10-truss in corresponding graphs.

All algorithms are implemented in C++ and compiled by GNU GCC 4.8.2 with -O2 optimization. For each test, we randomly generate 1000 queries and report the average processing

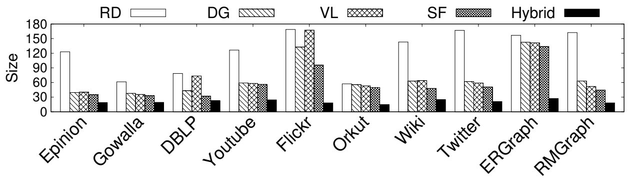


Fig. 3. Result Size with Different Vertex Selection Strategies

time. The time limit for the queries is 100s unless otherwise mentioned. All experiments are conducted on a machine with an Intel Xeon 2.2GHz CPU and 512GB main memory.

### B. *Evaluation of Effectiveness*

In this section, we firstly report the average size of triangle-connected $k$-truss computed by our algorithms and other four heuristic algorithms. Then we compare our algorithm with other representative community search algorithms. Case studies are shown on DBLP to further validate the effectiveness. Note that we only report the results returned by Hybrid since Hybrid has the highest success ratio.

**Exp-1: Result Size.** In this experiment, we compare the average result size of our proposed algorithm, Hybrid, with other heuristic algorithms. The results are reported in Figure 3. The size constraint for SCkT is 30. As shown in our experiment, among the heuristic algorithms, the heuristic algorithm SF using our score function has the best performance. This is because SF dynamically selects the promising vertices to be added based on current partial solution. In general, our proposed algorithm (SCkT) has the smallest result size as the size of SCkT is no larger than 30.

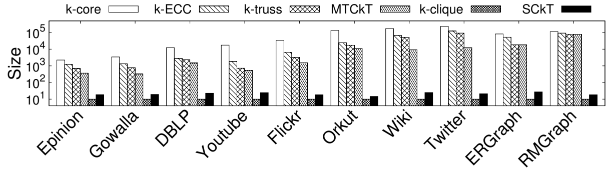


Fig. 4. Result Size with Different Community Search Methods

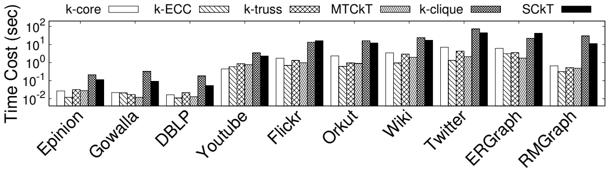


Fig. 5. Runtime with Different Community Search Methods

**Exp-2: Comparison with Other Methods.** In this experiment, we compare SCkT with five representative community search methods introduced in the survey of community search [17], i.e., $k$-core [5], $k$-ECC [8], $k$-truss [20], maximal triangle-connected $k$-truss (MTCkT) [1], and $k$-clique [44]. $k$ is set as 10 in all these algorithms. The above methods are index-based algorithms and the index construction time is excluded from runtime. For instance, we retrieve the resulting $k$-core from its index of core decomposition [5]. The compared methods will be much slower without the indexes. The SCkT is computed by Hybrid using default settings $s = 30$ and $k = 10$. For each graph, we evaluate the community size and two important scores: the clustering coefficient ($\frac{3\times|\triangle|}{|triplet|}$) [29] and graph density ($\frac{2m}{n\times(n-1)}$) [12].

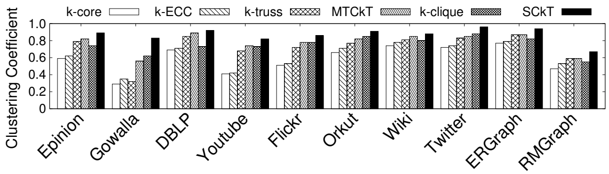


Fig. 6. Clustering Coefficient (The score of any method is counted as 0 if a query returns empty.)

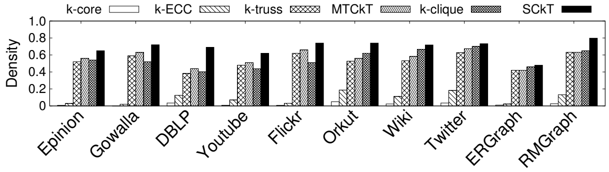
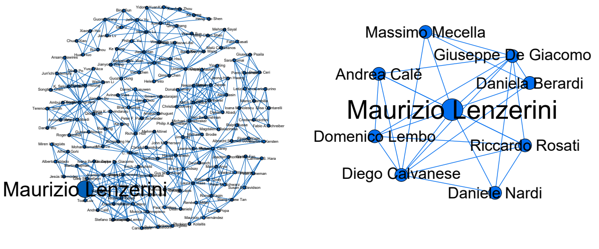


Fig. 7. Graph Density (The score of any method is counted as 0 if a query returns empty.)

As shown in Figure 4, compared with $k$-core, $k$-ECC, $k$-truss, and MTCkT which may return communities with tens of thousands vertices, our SCkT search algorithm returns much smaller communities within the size threshold. Due to the size constraint, SCkT has well-connected inner structures and achieves higher scores in terms of clustering coefficient (Figure 6) and graph density (Figure 7). Though the size of $k$-clique is only $k$, it often returns empty results due to the strict restriction. Furthermore, despite the NP-hardness of SCkT search problem, the runtime of SCkT is comparable with the methods without size constraint (Figure 5). Therefore, SCkT is an efficient model in finding communities with high quality.



(a) 5-truss (b) SCkT

Fig. 8. Case Study on DBLP

**Exp-3: Case Study on DBLP.** Figure 8 (a) depicts the entire 5-truss of the author "Maurizio Lenzerini" in DBLP. Figure 8 (b) is the SCkT of "Maurizio Lenzerini" computed by Hybrid with $k = 5$ and $s = 10$. In DBLP, there is an edge between two authors iff they have co-authored at least 3 papers. The entire connected 5-truss has 160 vertices, 534 edges, and density of 0.042 while the SCkT has 9 vertices, 26 edges, and density of 0.72. Compared with the entire connected 5-truss, our SCkT is denser as the SCkT does not include authors far away from "Maurizio Lenzerini". Moreover, we find that the average number of co-authored papers between each author in Figure 8 (a) is 0.6 while that value in Figure 8 (b) is 4.5, which shows that the relationship between each author in SCkT is closer.

TABLE III
SUCCESS RATIO OF QUERIES ($s = 30$ AND $k = 10$)

| Dataset | Success ratio (%) | | | | | |
|---|---|---|---|---|---|---|
| | Expand-D | Expand | Shrink-D | Shrink | Hybrid-D | Hybrid |
| Epinion | 54.2 | 77.8 | 68.2 | 99.1 | 67.4 | **99.3** |
| Gowalla | 81.3 | 95.5 | 76.3 | 98.2 | 83.4 | **99.0** |
| DBLP | 91.3 | 99.0 | 91.4 | 99.0 | 92.3 | **99.2** |
| YouTube | 65.3 | 74.7 | 87.3 | 97.0 | 85.9 | **98.3** |
| Flickr | 82.3 | 93.8 | 85.6 | 96.8 | 85.2 | **98.8** |
| Orkut | 66.2 | 98.2 | 72.1 | 98.0 | 75.3 | **98.7** |
| Wiki | 78.1 | 95.6 | 82.3 | 96.3 | 87.9 | **98.3** |
| Twitter | 81.4 | 87.6 | 80.2 | 93.2 | 85.7 | **96.3** |
| ERGraph | 56.3 | 88.0 | 62.1 | 86.5 | 74.1 | **92.4** |
| RMGraph | 87.2 | 97.3 | 86.7 | 96.9 | 90.8 | **99.2** |

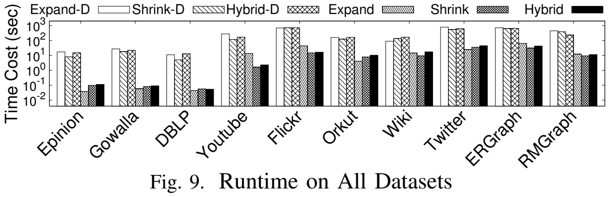


Fig. 9. Runtime on All Datasets

### C. Evaluation of Efficiency

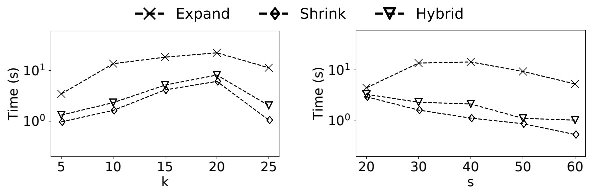
In this section, we test the runtime and success ratio of our proposed SCkT search algorithms on all the datasets. We also investigate the query time with different $k$ and $s$ values. At the end of this section, we report the runtime for multi-vertex queries. To avoid bias, the runtime is the average time of the queries returned within time limit.

**Exp-4: Runtime and Success Ratio on All Datasets.** In this experiment, we report the success ratio and average runtime of Expand, Shrink, Hybrid, Expand-D, Shrink-D, and Hybrid-D on all datasets. We lift the time limit to 1000s in this test, as the methods with IELB (D) cost much more time. Success ratio is the number of queries successfully returned before timeout over the total number of queries. The results are shown in Figure 9 and Table III. Note that a slight portion of hard cases cannot be computed within the time limit due to the huge search space from the NP-hardness.

In general, Hybrid outperforms Expand and Shrink in terms of success ratio. The reason is that Hybrid can switch between Expand and Shrink, and choose a suitable search strategy based on current partial solution. Compared with others, the runtime for Hybrid is not as good as success ratio. This is because some hard cases (timeout queries in Expand and Shrink) are returned in Hybrid which lifts the average time. Nevertheless, the runtime for Hybrid is still comparable with other algorithms, i.e., it is efficient in practice.

TABLE IV
SUCCESS RATIO WHEN VARYING $k$ AND $s$.

| vary $k$ ($s = 30$) | Youtube | | | vary $s$ ($k = 10$) | Youtube | | |
|---|---|---|---|---|---|---|---|
| | Expand | Shrink | Hybrid | | Expand | Shrink | Hybrid |
| 5 | 95.9 | 98.1 | **98.4** | 20 | 82.2 | 97.0 | **98.5** |
| 10 | 74.7 | 97.0 | **98.3** | 30 | 74.7 | 97.0 | **98.3** |
| 15 | 70.5 | 93.2 | **95.0** | 40 | 88.3 | 98.4 | **99.1** |
| 20 | 89.0 | 94.2 | **95.1** | 50 | 96.9 | 99.0 | **99.2** |
| 25 | 97.7 | **98.9** | **98.9** | 60 | **99.2** | **99.2** | **99.2** |



(a) Youtube ($s = 30$)

(b) Youtube ($k = 10$)

Fig. 10. Query Time when Varying $k$ and $s$

**Exp-5: Evaluation of Varying $k$.** In this experiment, we report the average runtime and success ratio of Expand, Shrink, and Hybrid with $k$ varying from 5 to 25 and $s = 30$. We only report the results on Youtube since the trends are similar in other datasets. The results are shown in Figure 10 (a) and the left part of Table IV. As $k$ increases, we find that the runtime of all three algorithms firstly rises up then drops down. The reason is that the runtime with varying $k$ is affected by two factors. First, as $k$ increases, we need to explore more vertices as it becomes more difficult to form a triangle-connected $k$-truss when $s$ is fixed. Second, at the same time, the search space is reduced due to less candidate vertices. As observed on Youtube, when $k$ is relatively small, the first factor makes it harder to find a SCkT. Therefore, the runtime rises up and success ratio drops down. Then, the second factor gains more influence and the runtime drops down while success ratio rises up. Hybrid has the highest success ratio and small time cost.

**Exp-6: Evaluation of Varying $s$.** In this experiment, we report the average runtime and success ratio of Expand, Shrink, and Hybrid on Youtube with $s$ varying from 20 to 60 and $k = 10$. The results are shown in Figure 10 (b) and the right part of Table IV. The runtime of Shrink decreases and the success ratio rises up with the increasing of $s$. This is because as $s$ increases, it becomes easier to find a SCkT. On the other hand, as analyzed in Section IV-D, the time complexity of Shrink reduces with the increasing of $s$ when $k$ is fixed. The runtime and success ratio of Expand are similar to the results in Exp-5. The reason is that as $s$ increases, the search space of Expand increases. At the same time, it becomes easier to find a solution. When $s$ is large enough, Expand can quickly expand a SCkT in most cases. For Hybrid, its success ratio outperforms other algorithms and its time cost is efficient.

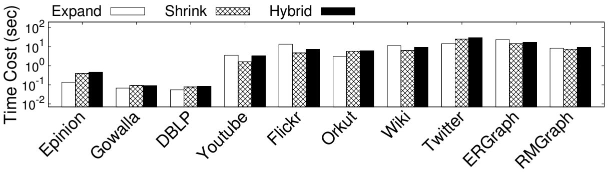


Fig. 11. Multiple-vertex Queries on All Datasets

**Exp-7: Multi-vertex Queries on All Datasets.** In this experiment, we report the average time cost for multi-vertex query of Expand, Shrink, and Hybrid on all datasets. For each query, we randomly select five vertices. The results are reported in Figure 11. As shown in our experiment, all the three algorithms have comparable performance on runtime. The success ratio, which is omitted due to the space limitation, is similar to that of single vertex query and Hybrid achieves the highest success ratio on all the datasets.

## VI. CONCLUSION

In this paper, we study the SCkT search problem which aims to find a triangle-connected $k$-truss containing the query vertices with size not exceeding a threshold. We prove the problem is NP-hard. A practically-efficient exact solution is developed which employs a novel lower bound and effective search strategies. Extensive experiments verify the effectiveness of SCkT and the superiority of the techniques. As a future study, we may explore approximate solutions to address the extreme cases caused by the hardness of the problem.

## Acknowledgments

Fan Zhang is supported by NSFC62002073. Xuemin Lin is supported by 2018YFB1003504, NSFC61232006, ARC DP180103096 and DP170101628. Wenjie Zhang is supported by ARC DP180103096. Ying Zhang is supported by ARC DP180103096 and FT170100128.

## References

[1] E. Akbas and P. Zhao. Truss-based community search: a truss-equivalence based indexing approach. *PVLDB*, 10(11):1298–1309, 2017.
[2] D. A. Bader and K. Madduri. Gtgraph: A synthetic graph generator suite. *Atlanta, GA, February*, 38, 2006.
[3] A.-L. Barabasi and Z. N. Oltvai. Network biology: understanding the cell's functional organization. *Nature reviews genetics*, 5(2):101, 2004.
[4] N. Barbieri, F. Bonchi, E. Galimberti, and F. Gullo. Efficient and effective community search. *Data Mining and Knowledge Discovery*, 29(5):1406–1433, Sep 2015.
[5] V. Batagelj and M. Zaversnik. An o(m) algorithm for cores decomposition of networks. *CoRR*, cs.DS/0310049, 2003.
[6] V. Batagelj and M. Zaveršnik. Short cycle connectivity. *Discrete Mathematics*, 307(3-5):310–318, 2007.
[7] D. Chakrabarti, Y. Zhan, and C. Faloutsos. R-mat: A recursive model for graph mining. In *ICDM*, pages 442–446. SIAM, 2004.
[8] L. Chang, X. Lin, L. Qin, J. X. Yu, and W. Zhang. Index-based optimal algorithms for computing steiner components with maximum connectivity. In *SIGMOD*, pages 459–474, 2015.
[9] X. Chen, L. Lai, L. Qin, and X. Lin. Structsim: Querying structural node similarity at billion scale. In *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020*, pages 1950–1953, 2020.
[10] X. Chen, L. Lai, L. Qin, X. Lin, and B. Liu. A framework to quantify approximate simulation on graph data. *arXiv preprint arXiv:2010.08938*, 2020.
[11] J. Cohen. Trusses: Cohesive subgraphs for social network analysis. *National security agency technical report*, 16:3–1, 2008.
[12] T. F. Coleman and J. J. Moré. Estimation of sparse jacobian matrices and graph coloring blems. *SIAM journal on Numerical Analysis*, 20(1):187–209, 1983.
[13] W. Cui, Y. Xiao, H. Wang, and W. Wang. Local search of communities in large graphs. In *SIGMOD*, pages 991–1002, 2014.
[14] M. T. Dittrich, G. W. Klau, A. Rosenwald, T. Dandekar, and T. Müller. Identifying functional modules in protein–protein interaction networks: an integrated exact approach. *Bioinformatics*, 24(13):i223–i231, 2008.
[15] P. Erdős and A. Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.
[16] Y. Fang, R. Cheng, S. Luo, and J. Hu. Effective community search for large attributed graphs. *PVLDB*, 9(12):1233–1244, 2016.
[17] Y. Fang, X. Huang, L. Qin, Y. Zhang, W. Zhang, R. Cheng, and X. Lin. A survey of community search over big graphs. *VLDB Journal*, 2019.
[18] C. Giatsidis, D. M. Thilikos, and M. Vazirgiannis. D-cores: Measuring collaboration of directed graphs based on degeneracy. In *ICDM*, pages 201–210, 2011.
[19] J. Hu, X. Wu, R. Cheng, S. Luo, and Y. Fang. On minimal steiner maximum-connected subgraphs in large graphs. *TKDE*, 29(1):2455–2469, 2017.
[20] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu. Querying k-truss community in large and dynamic graphs. In *SIGMOD*, pages 1311–1322, 2014.
[21] X. Huang and L. V. S. Lakshmanan. Attribute-driven community search. *PVLDB*, 10(9):949–960, 2017.
[22] X. Huang, L. V. S. Lakshmanan, J. X. Yu, and H. Cheng. Approximate closest community search in networks. *PVLDB*, 9(4):276–287, 2015.
[23] X. Huang, W. Lu, and L. V. S. Lakshmanan. Truss decomposition of probabilistic graphs: Semantics and algorithms. In *SIGMOD*, pages 77–90, 2016.
[24] C. Li, F. Zhang, Y. Zhang, L. Qin, W. Zhang, and X. Lin. Efficient progressive minimum k-core search. *PVLDB*, 13(3):362–375, 2019.
[25] B. Liu, L. Yuan, X. Lin, L. Qin, W. Zhang, and J. Zhou. Efficient ($\alpha$, $\beta$)-core computation: an index-based approach. In *The World Wide Web Conference*, pages 1130–1141. ACM, 2019.
[26] B. Liu, L. Yuan, X. Lin, L. Qin, W. Zhang, and J. Zhou. Efficient ($\alpha$, $\beta$)-core computation in bipartite graphs. *VLDB J.*, 29(5):1075–1099, 2020.
[27] B. Liu, F. Zhang, C. Zhang, W. Zhang, and X. Lin. Corecube: Core decomposition in multilayer graphs. In *International Conference on Web Information Systems Engineering*, volume 11881, pages 694–710. Springer, 2019.
[28] Q. Liu, M. Zhao, X. Huang, J. Xu, and Y. Gao. Truss-based community search over large directed graphs. In *SIGMOD*, pages 2183–2197, 2020.
[29] R. D. Luce and A. D. Perry. A method of matrix analysis of group structure. *Psychometrika*, 14(2):95–116, 1949.
[30] Y. Ma, Y. Yuan, F. Zhu, G. Wang, J. Xiao, and J. Wang. Who should be invited to my party: A size-constraint k-core problem in social networks. *J. Comput. Sci. Technol*, 34(1):170–184, 2019.
[31] G. Palla, I. Derényi, I. Farkas, and T. Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *nature*, 435(7043):814, 2005.
[32] Y. Peng, Y. Zhang, W. Zhang, X. Lin, and L. Qin. Efficient probabilistic k-core computation on uncertain graphs. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 1192–1203. IEEE, 2018.
[33] A. E. Sariyüce and A. Pinar. Fast hierarchy construction for dense subgraphs. *PVLDB*, 10(3):97–108, 2016.
[34] A. E. Sariyüce, C. Seshadhri, A. Pinar, and Ü. V. Catalyürek. Finding the hierarchy of dense subgraphs using nucleus decompositions. In *WWW*, pages 927–937, 2015.
[35] R. Sharan, I. Ulitsky, and R. Shamir. Network-based prediction of protein function. *Molecular systems biology*, 3(1), 2007.
[36] M. Sozio and A. Gionis. The community-search problem and how to plan a successful cocktail party. In *SIGKDD*, pages 939–948, 2010.
[37] J. Ugander, L. Backstrom, C. Marlow, and J. M. Kleinberg. Structural diversity in social contagion. *Proc. Natl. Acad. Sci. USA*, 109(16):5962–5966, 2012.
[38] J. Wang and J. Cheng. Truss decomposition in massive networks. *PVLDB*, 5(9):812–823, 2012.
[39] K. Wang, X. Cao, X. Lin, W. Zhang, and L. Qin. Efficient computing of radius-bounded k-cores. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 233–244. IEEE, 2018.
[40] K. Wang, X. Lin, L. Qin, W. Zhang, and Y. Zhang. Vertex priority based butterfly counting for large-scale bipartite networks. *Proceedings of the VLDB Endowment*, 12(10):1139–1152, 2019.
[41] K. Wang, W. Zhang, X. Lin, Y. Zhang, L. Qin, and Y. Zhang. Efficient and effective community search on large-scale bipartite graphs. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2021.
[42] S. Wasserman, K. Faust, et al. *Social network analysis: Methods and applications*, volume 8. Cambridge university press, 1994.
[43] D. J. Watts and S. H. Strogatz. Collective dynamics of small-world networks. *Nature*, 393(6684):440, 1998.
[44] L. Yuan, L. Qin, W. Zhang, L. Chang, and J. Yang. Index-based densest clique percolation community search in networks. *TKDE*, 30(5):922–935, 2018.
[45] C. Zhang, F. Zhang, W. Zhang, B. Liu, Y. Zhang, L. Qin, and X. Lin. Exploring finer granularity within the cores: Efficient (k, p)-core computation. In *ICDE*, pages 181–192. IEEE, 2020.
[46] F. Zhang, W. Zhang, Y. Zhang, L. Qin, and X. Lin. Olak: an efficient algorithm to prevent unraveling in social networks. *Proceedings of the VLDB Endowment*, 10(6):649–660, 2017.
[47] F. Zhang, Y. Zhang, L. Qin, W. Zhang, and X. Lin. Finding critical users for social network engagement: The collapsed k-core problem. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, pages 245–251, 2017.
[48] F. Zhang, Y. Zhang, L. Qin, W. Zhang, and X. Lin. Efficiently reinforcing social networks over user engagement and tie strength. In *ICDE*, pages 557–568, 2018.
[49] J. Zhang, P. S. Yu, and Y. Lv. Enterprise employee training via project team formation. In *WSDM*, pages 3–12, 2017.
[50] Z. Zheng, F. Ye, R.-H. Li, G. Ling, and T. Jin. Finding weighted k-truss communities in large networks. *Information Sciences*, 417:344–360, 2017.