Truss-Based Structural Diversity Search in Large Graphs

Jinbin Huang, Xin Huang, and Jianliang Xu

Abstract—Social decisions made by individuals are easily influenced by information from their social neighborhoods. A key predictor of social contagion is the multiplicity of social contexts inside the individual's contact neighborhood, which is termed structural diversity. However, the existing models have limited decomposability for analyzing large-scale networks, and suffer from the inaccurate reflection of social context diversity. In this paper, we propose a truss-based structural diversity model to overcome the weak decomposability. Based on this model, we study a novel problem of truss-based structural diversity search in a graph G, that is, to find the r vertices with the highest truss-based structural diversity and return their social contexts. To tackle this problem, we propose an online structural diversity search algorithm in $O(\rho(m+\mathcal{T}))$ time, where ρ, m, and $\mathcal{T}$ are respectively the arboricity, the number of edges, and the number of triangles in G. To improve the efficiency, we design an elegant and compact index, called TSD-index, which keeps the structural diversity information for all individual vertices. We further optimize the structure of TSD-index into a highly compressed GCT-index. Our GCT-index-based structural diversity search utilizes the global triangle information for fast index construction and finds answers in $O(m)$ time. Extensive experiments demonstrate the effectiveness and efficiency of our proposed model and algorithms, against state-of-the-art methods.

Index Terms—Structural diversity, top-k search, social contagion, truss mining

1 Introduction

Online social networks (Twitter, Facebook, Instagram, etc.) have been important platforms for individuals to exchange information with their friends. Social contagion [5], [24], [28], [35] is a phenomenon that individuals are influenced by the information received from their social neighborhoods, e.g., acting the same as friends in sharing posts or adopting political opinions. Social decisions made by individuals often depend on the *multiplicity of distinct social contexts* inside his/her contact neighborhood, which is termed *structural diversity* [6], [19], [35]. Many studies on Facebook [35] show that users are much more likely to join Facebook and become engaged if they have a larger structural diversity, i.e., a larger number of distinct social contexts. Given the important role of structural diversity, a fundamental problem of structural diversity search is to find the r users with the highest structural diversity in graphs [6], [19], which can be beneficial to political campaigns [23], viral marketing [24], promotion of health practices [35], cooperation in social dilemmas [29], and so on.

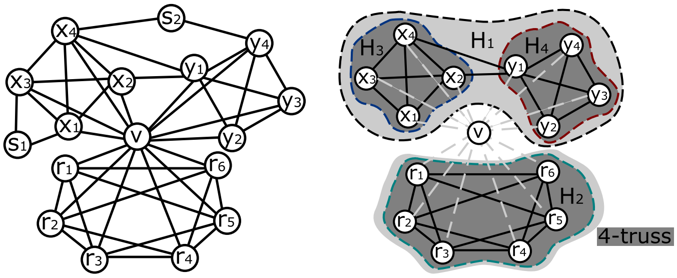
The problem of structural diversity search has been recently studied based on two structural diversity models of k-sized component [6], [19] and k-core [18]. However, one significant limitation of both models is their limited decomposability for analyzing large-scale networks, which may lead to inaccurate reflection of social context diversity. The detailed quality analysis and case studies can be found in Sections 7.2 and 7.3. To address this issue, in this paper, we propose a new structural diversity model based on k-truss. A k-truss requires that every edge is contained in at least (k-2) triangles in the k-truss [9]. Intuitively, a k-truss signifies strong social ties among the members in this social group, while tending to break up weak-tied social groups and discard tree-like components. Our model treats each maximal connected k-truss as a distinct social context. As we will demonstrate, our model has several major advantages. First, thanks to k-truss, our model has a strong decomposability for analyzing large-scale networks at different levels of granularity. Second, a compact and elegant index can be designed for efficient truss-based structural diversity search in a linear cost w.r.t. graph size. Third, when compared with other models, our model shows superiority in the evaluation of influence propagation on real-world networks.

Motivating Example. Consider a social network G in Fig. 1a. The ego-network of an individual v is a subgraph of G formed by all v's neighbors as shown in the light gray region (excluding vertex v) in Fig. 1b. To analyze the social contexts in Fig. 1b, different structural diversity models have substantial differences:

- *Component-based structural diversity model* regards each connected component of vertex size at least k as a social context [6], [19]. The component H_1 having 8 vertices is regarded as one social context. However, in terms of graph structure, two subgraphs H_3 and H_4 shown in Fig. 1b are loosely connected through edges (x_2, y_1) and (x_4, y_1), and vertices (x_1 and x_3) span long distances to vertices (y_2, y_3 and y_4). Thus, H_3 and H_4 can be reasonably treated as two different

- *The authors are with the Hong Kong Baptist University, Kowloon Tong, Hong Kong. E-mail: {jbhuang, xinhuang, xujl}@comp.hkbu.edu.hk.*

Manuscript received 16 June 2019; revised 10 July 2020; accepted 13 Sept. 2020. Date of publication 30 Sept. 2020; date of current version 7 July 2022.
(Corresponding author: Xin Huang.)
Recommended for acceptance by N. Zhang.
Digital Object Identifier no. 10.1109/TKDE.2020.3027950



(a) Graph G (b) Ego-Network $G_{N(v)}$

Fig. 1. A running example.

social contexts. Unfortunately, the attempt of adjusting parameter k *using any value* does not help the decomposition of H_1.

- *Core-based structural diversity model* regards a maximal connected k-core as a social context [18], [35]. A k-core requires that every vertex has degree at least k within the k-core. For $1 \leq k \leq 3$, H_1 is regarded as one maximal connected k-core, which cannot be decomposed into disjoint components; for $k \geq 4$, H_1 is no longer counted as a feasible social context.
- *Our truss-based structural diversity model* treats each maximal connected k-truss as a distinct social context. For $k = 4$, H_1 is decomposed into two maximal connected 4-trusses H_3 and H_4 in Fig. 1b, where each edge has at least two triangles. As a result, H_2, H_3 and H_4 are regarded as three distinct social contexts in the ego-network of v, and the structural diversity of v is 3.

In light of the above example, truss-based structural diversity search is a pressing need. However, to the best of our knowledge, the problem of truss-based structural diversity search over graphs, has not been studied yet. In this paper, we investigate the problem to find the r vertices with the highest truss-based structural diversity and return their social contexts. We propose efficient algorithms for truss-based structural diversity search.

However, efficient computation of truss-based structural diversity search raises significant challenges. A straightforward online search algorithm is to compute the structural diversity for all vertices and return the top-r vertices, which is inefficient. Because it is costly to compute the structural diversity for all vertices in large graphs, from scratch without any pruning. The subgraph extraction of an ego-network needs the costly operation of triangle listing [25], not even talking about the truss decomposition [36] for finding all maximal connected k-trusses. On the other hand, developing a diversity bound for pruning search space is also difficult. Unlike the symmetry structure of ego-networks in the component-based model [6], [19], non-symmetry structural properties restrict our truss-based model to derive an efficient pruning bound. Therefore, existing structural diversity algorithms for component-based and core-based models [6], [18], [19] do not work for our truss-based model.

Fortunately, truss-based structural diversity has many desirable features for developing efficient indexes and algorithms. To improve the efficiency of truss-based structural diversity search, we propose several useful optimization techniques. We develop an efficient top-r search framework to prune vertices for avoiding structural diversity computation. The heart of our framework is to exploit two important pruning techniques: (1) graph sparsification and (2) a diversity bound. Specifically, we first make use of structural properties of k-truss and propose graph sparsification to remove from the graph unqualified edges and nodes that will not be in any k-truss. Second, we develop an upper bound of diversity for pruning unqualified answers, leading to an early termination of our top-r search. Furthermore, we develop a novel truss-based structural diversity index, called TSD-index, which is a compact and elegant tree structure to keep the structural information for all ego-networks in G. Based on the TSD-index, we propose an index-based top-r search algorithm to quickly find answers. Furthermore, to explore the sharing computation across vertices, we utilize the global triangle listing one-shot for fast ego-network extraction and develop a fast bitmap technique for ego-network decomposition. Leveraging a new data structure of GCT-index compressed from TSD-index, we propose GCT for truss-based structural diversity search, which achieves a smaller index size and a faster query time.

To summarize, we make the following contributions:

- We use a maximal connected k-truss to model a neighborhood social context in the ego-network. We define the truss-based structural diversity and then formulate a new problem of truss-based structural diversity search over graphs. (Section 2)
- We present a method of computing truss-based structural diversity using truss decomposition. Based on this, we develop an online search algorithm to tackle our problem, and give a comprehensive theoretical analysis of algorithm complexity. (Section 3)
- We analyze the structural properties of truss-based social contexts, and develop two useful pruning techniques of graph sparsification and a diversity bound. Equipped with them, we develop an efficient framework for structural diversity search with an early termination mechanism. (Section 4)
- We design a space-efficient truss-based structural diversity index (TSD-index) to keep the structural diversity information for all ego-networks. We propose a TSD-index-based search algorithm to quickly find answers in a linear cost w.r.t. graph size. (Section 5)
- We propose GCT for truss-based structural diversity search based on the efficient techniques of fast ego-network truss decomposition and a compressed GCT-index. (Section 6)
- We validate the efficiency and effectiveness of our methods through extensive experiments. (Section 7)

We discuss related work in Section 8, and conclude the paper with a summary in Section 9.

2 Problem Definition

We consider an undirected and unweighted simple graph $G = (V, E)$ with $n = |V|$ vertices and $m = |E|$ edges. We define $N(v) = \{u \in V : (v, u) \in E\}$ as the set of neighbors of a vertex v, and $d(v) = |N(v)|$ as the degree of v in G. Let d_{max} represent the maximum degree in G. For a set of vertices $S \subseteq V$, the induced subgraph of G by S is denoted by G_S,

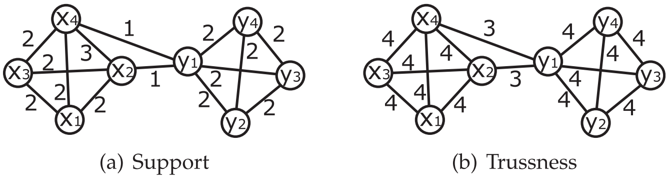


Fig. 2. The support and trussness of edges in H_1.

where the vertex set is $V(G_S) = S$ and the edge set is $E(G_S) = \{(v, u) \in E : v, u \in S\}$. W.l.o.g. we assume that the considered graph G is connected, indicating that $m \geq n - 1$ and $n \in O(m)$. The assumption is similarly made in [18], [25].

2.1 Ego-Network

We define an ego-network [11], [27] in the following.

Definition 1 [Ego-Network]. *Given a vertex $v \in V$, the ego-network of v, is a subgraph of G induced by the vertex set $N(v)$, denoted by $G_{N(v)}$, where the vertex set $V(G_{N(v)}) = N(v)$ and the edge set $E(G_{N(v)}) = \{(u, w) \in E : u, w \in N(v)\}$.*

In the literature, the term "neighborhood induced subgraph of v" [18] has also been used to indicate the ego-network of v, since the ego-network is formed by all neighbors of v. For example, consider the graph G in Fig. 1a and the vertex $v \in V$, the ego-network of v is shown as the gray region in Fig. 1b, which is formed by the induced subgraph of G by vertices $N(v) = \{x_1, \ldots, x_4, y_1, \ldots, y_4, r_1, \ldots, r_6\}$, excluding the center vertex v with its incident edges.

2.2 Truss-Based Structural Diversity

A triangle in G is a cycle of length 3. Given three vertices $u, v, w \in V$, the triangle formed by u, v, w is denoted by $\triangle_{uvw}$. Given a subgraph $H \subseteq G$, the support of an edge $e = (u, v) \in E(H)$ is defined as the number of triangles containing edge e in H, i.e., $\sup_H(e) = |\{\triangle_{uvw} : (u, w), (v, w) \in E(H)\}|$. Fig. 2a shows the support of each edge in graph H_1. There exists only one triangle $\triangle_{x_2 x_4 y_1}$ containing (x_2, y_1), and $\sup_{H_1}(x_2, y_1) = 1$. We drop the subscript and denote the support as $\sup(e)$, when the context is obvious.

A k-truss of graph G is defined as the largest subgraph of G such that every edge has support of at least $k - 2$ in this subgraph [20], [36]. For a given $k \geq 2$, the k-truss of a graph G is unique, which may be disconnected with multiple components. In our truss-based structural diversity model, we treat each connected component of the k-truss as a distinct *social context*. The definition of social contexts in an ego-network is given below.

Definition 2 (Social Contexts). *Given a vertex v and an integer $k \geq 2$, each connected component of the k-truss in $G_{N(v)}$ is called a social context. Thus, the social contexts of v are represented by all vertex sets of components, denoted by $\mathsf{SC}(v) = \{V(H) : H \text{ is a connected component of the } k\text{-truss in } G_{N(v)}\}$.*

By Definition 2, each social context is a component of k-truss, which is connected and also the maximal subgraph of the k-truss. Therefore, as an alternative, we also call a social context as a *maximal connected k-truss* throughout the paper. For example, consider an ego-network $G_{N(v)}$ in Fig. 1b and $k = 4$. The 4-truss of $G_{N(v)}$ is presented by the darker gray region. We regard a connected component H_3 as a neighborhood social context in $G_{N(v)}$, which is represented by $V(H_3) = \{x_1, x_2, x_3, x_4\}$. Thus, the social contexts of v have $\mathsf{SC}(v) = \{\{x_1, x_2, \ x_3, x_4\}, \{y_1, y_2, y_3, y_4\}, \{r_1, r_2, r_3, r_4, r_5, r_6\}\}$.

Based on the definition of social contexts, we can define our key concept of *truss-based structural diversity* as follows.

Definition 3 (Truss-based Structural Diversity). *Given a vertex v and an integer $k \geq 2$, the truss-based structural diversity of v is the multiplicity of social contexts $\mathsf{SC}(v)$, denoted by $score(v) = |\mathsf{SC}(v)|$.*

The truss-based structural diversity is exactly the number of connected components of the k-trusses in the ego-network. Consider the ego-network $G_{N(v)}$ in Fig. 1b and $k = 4$, the 4-truss of $G_{N(v)}$ has three connected components H_2, H_3, and H_4, thus $\mathsf{score}(v) = 3$.

2.3 Problem Statement

The problem of truss-based structural diversity search studied in this paper is formulated as follows.

Problem Statement. Given a graph G and two integers r and k where $1 \leq r \leq n$ and $k \geq 2$, the goal of top-r truss-based structural diversity search is to find a set of r vertices in G having the highest scores of truss-based structural diversity w.r.t. the trussness threshold k, and return their social contexts.

Consider the graph G in Fig. 1 with $r = 1$ and $k = 4$, the answer of our problem is the vertex v, which has the highest structural diversity $\mathsf{score}(v) = 3$ and its social contexts $\mathsf{SC}(v) = \{\{x_1, x_2, x_3, x_4\}, \{y_1, y_2, y_3, y_4\}, \{r_1, r_2, r_3, r_4, r_5, r_6\}\}$.

3 Online Search Algorithm

In this section, we develop an online search algorithm for top-r truss-based structural diversity search and analyze the algorithm complexity.

3.1 Truss Decomposition

Trussness. We start with a useful definition of trussness below.

Definition 4 (Trussness). *Given a subgraph $H \subseteq G$, the trussness of H is defined as the minimum support of edges in H plus 2, denoted by $\tau(H) = \min_{e \in E(H)} \{\sup_H(e) + 2\}$. The trussness of an edge $e \in H$ denoted by $\tau_H(e)$ is defined as the largest number k such that there exists a connected k-truss $H' \subseteq H$ containing e, i.e.,*

$$\tau_H(e) = \max_{H' \subseteq H, e \in E(H')} \tau(H').$$

Similar to the notation of support, we drop the subscript and denote the trussness $\tau_H(e)$ as $\tau(e)$ when the context is obvious. In addition, we define the trussness of a vertex v as $\tau_H(v) = \max_{H' \subseteq H, v \in V(H')} \tau(H')$.

Example 1. Fig. 2b shows the trussness of each edge in graph H_1. First, according to the edge support in Fig. 2a, the trussness of subgraph H_1 is $\tau(H_1) = \min_{e \in E(H_1)} \{\sup_{H_1}(e) + 2\} = 1 + 2 = 3$. Thus, we have $\tau_{H_1}(x_2, y_1) = \max_{H' \subseteq H_1, e \in E(H')} \tau(H') = 3$.

Algorithm of Truss Decomposition. Truss decomposition on graph G is to find the k-trusses of G for all possible k's and compute the trussnesses of all edges in G. In this paper, we adopt the truss decomposition algorithm proposed in [36]. The algorithm keeps peeling the edges with the smallest support in the remaining graph to obtain the trussness of each edge. We omit the detail of the algorithm for brevity.

3.2 Computing $score(v)$

Algorithm 1 presents a procedure of computing $score(v)$, which calculates the number of maximal connected k-trusses in the ego-network $G_{N(v)}$. The algorithm first extracts $G_{N(v)}$ from graph G (line 1), and then applies the truss decomposition in [36] on $G_{N(v)}$ (line 2). After obtaining the trussness of all edges, it removes all the edges e with $\tau_{G_{N(v)}}(e) < k$ from $G_{N(v)}$ (line 3). The remaining graph $G_{N(v)}$ is the union of all maximal connected k-trusses. Applying the breadth-first-search, all connected components are identified as the social contexts $\mathsf{SC}(v) = \{V(H) : H$ is a maximal connected k-truss in $G_{N(v)}\}$ (line 4). Algorithm 1 finally returns the structural diversity $score(v) = |\mathsf{SC}(v)|$ (lines 5-6).

Algorithm 1. Computing $score(v)$

```
Input: $G = (V, E)$, a vertex $v$, the trussness threshold $k$
Output: $score(v)$
1: Extract an ego-network of $v$ as $G_{N(v)}$ from $G$ by Definition 1;
2: Apply the truss decomposition on $G_{N(v)}$;
3: Remove all edges $e$ with $\tau_{G_{N(v)}}(e) < k$ from $G_{N(v)}$;
4: Identify all connected components in $G_{N(v)}$ as the social contexts $\mathsf{SC}(v) = \{V(H) : H$ is a maximal connected $k$-truss in $G_{N(v)}\}$;
5: $score(v) \leftarrow |\mathsf{SC}(v)|$;
6: return $score(v)$;
```

3.3 Online Search Algorithm

Equipped with the procedure of computing $score(v)$, we present an online search algorithm to address the problem of top-r structural diversity search. The online search algorithm computes the structural diversity for all vertices in graph G from scratch with respect to a pair of parameters k and r. It maintains an answer set for recording the top-r results and returns the top-r results after the structural diversity for all vertices are computed.

3.4 Complexity Analysis

Lemma 1. *Algorithm 1 computes score(v) for v in* $O(\sum_{u\in N(v)} \min\{d(u), d(v)\} + \sum_{(u,w)\in E(G_{N(v)})} \min\{d(u), d(w)\})$ *time and* $O(m)$ *space.*

Proof. Because of space limitation, the detailed proof is reported in the arXiv article [17]. □

Theorem 1. *The online search algorithm runs on graph G taking*

$$O\left(\sum_{v\in V}\left\{\sum_{u\in N(v)} \min\{d(u), d(v)\} + \sum_{(u,w)\in E(G_{N(v)})} \min\{d(u), d(w)\}\right\}\right),$$

time and $O(m)$ *space.*

Proof. The online search algorithm uses Algorithm 1 to compute $score(v)$ for each vertex $v \in V$, which totally takes $O(\sum_{v\in V}\{\sum_{u\in N(v)} \min\{d(u), d(v)\} + \sum_{(u,w)\in E(G_{N(v)})} \min\{d(u), d(w)\}\})$ time by Lemma 1. Moreover the top-r results can be maintained in $O(n)$ time and $O(n)$ space, using bin sort. As a result, the online search algorithm takes $O(\sum_{v\in V}\{\sum_{u\in N(v)} \min\{d(u), d(v)\} + \sum_{(u,w)\in E(G_{N(v)})} \min\{d(u), d(w)\}\})$ time and $O(m+n) \subseteq O(m)$ space. □

Complexity Simplification. Theorem 1 has a tight time complexity, but in a very complex form. We relax the time complexity to simplify form using graph arboricity [8]. Specifically, the arboricity ρ of a graph G is defined as the minimum number of spanning trees that cover all edges of graph G, and $\rho \leq \min\{\lfloor\sqrt{m}\rfloor, d_{max}\}$ [8]. For any subgraph $g \subseteq G$, the arboricity ρ_g of g has $\rho_g \leq \rho$. We have the following theorem.

Theorem 2. *The online search algorithm runs on graph G taking* $O(\rho(m + \mathcal{T}))$ *time and* $O(m)$ *space, where* ρ *is the arboricity of G and* $\mathcal{T}$ *is the number of triangles in G.*

Proof. According to [8], $O(\sum_{(u,w)\in E(G)} \min\ \{d(u), d(v)\}) \subseteq O(\rho m)$, where ρ is the arboricity of G. Thus, we have

$$O\left(\sum_{v\in V}\left\{\sum_{u\in N(v)} \min\{d(u), d(v)\}\right\}\right)$$

$$\subseteq O\left(\sum_{(v,u)\in E} \min\{d(v), d(u)\}\right) \subseteq O(\rho m).$$

Now, we consider the remaining part of time complexity in Theorem 1 using the arboricity of ego-networks. For a vertex $v \in V$, the ego-network $G_{N(v)}$ has n_v vertices and m_v edges, where $n_v = |N(v)|$ and $m_v = |\{\triangle_{vuw} : u, w \in N(v), (u,w) \in E\}|$. Let the number of triangles in graph G be $\mathcal{T}$, and obviously $\mathcal{T} = \frac{\sum_{v\in V} m_v}{3}$. In addition, as $G_{N(v)} \subseteq G$, the arboricity ρ_v of $G_{N(v)}$ has $\rho_v \leq \rho$. As a result, we have

$$O\left(\sum_{v\in V}\left\{\sum_{(u,w)\in E(G_{N(v)})} \min\{d(u), d(w)\}\right\}\right)$$

$$\subseteq O\left(\sum_{v\in V} \rho_v m_v\right) \subseteq O\left(\rho \cdot \sum_{v\in V} m_v\right) \subseteq O(\rho\mathcal{T}).$$

Combining the above two equations, we have

$$O\left(\sum_{v\in V}\left\{\sum_{u\in N(v)} \min\{d(u), d(v)\} + \sum_{(u,w)\in E(G_{N(v)})} \min\{d(u), d(w)\}\right\}\right)$$

$$\subseteq O(\rho(m + \mathcal{T})).$$

□

4 An Efficient Top-r Search Framework

The online search algorithm is inefficient for top-r search, because it computes the structural diversity for all vertices on the entire graph. To improve the efficiency, we develop an efficient top-r search framework in this section. The heart of our framework is to exploit two important pruning techniques: (1) graph sparsification and (2) upper bounding $score(v)$.

4.1 Graph Sparsification

The goal of graph sparsification is to remove from graph G the unnecessary vertices and edges, which are not included in the maximal connected k-truss for any ego-network. This removal does not affect the answer, but shrinks the graph size for efficiency improvement.

*Structural Properties of k-Truss.*We start from a structural property of k-truss.

Property 1. *Given an edge $e^* \in E$, if $\tau_G(e^*) < (k+1)$, e^* will not be included in any maximal connected k-truss in the ego-network $G_{N(v)}$ for any vertex $v \in V$.*

Proof. We obmit the proof for brevity. The detailed proof can be found in the arXiv article [17]. □

Based on Property 1, we can safely remove any edge e with $\tau_G(e) < (k+1)$ and the isolated vertices from graph G after applying the truss decomposition [36] on G. Graph sparsification is a useful preprocessing step, which benefits efficiency improvement by reducing the graph size and avoiding the structural diversity computation of the isolated nodes.

4.2 An Upper Bound of $score(v)$

In this section, we analyze the structural properties of ego-networks and develop a tight upper bound of $score(v)$. Symmetry structure of ego-networks lends themselves to derive an efficient upper bound of structural diversity [6], [19]. However, the same symmetry properties fails in our truss-based structural diversity model. The following observation formalizes the property of non-symmetry.

Non-Symmetry. Consider three vertices u, v, w form a triangle $\triangle_{uvw}$ in G. The non-symmetry of truss-based structural diversity shows that the edges (v,w), (u,w), (u,v) may have different trussnesses in the ego-networks $G_{N(u)}$, $G_{N(v)}$, $G_{N(w)}$ respectively. In other words, $\tau_{G_{N(u)}}(v,w)$, $\tau_{G_{N(v)}}(u,w)$, and $\tau_{G_{N(w)}}(u,v)$ may not be the same. For example, we consider three vertices v, r_1, and r_2 in graph G shown in Fig. 1a. For ego-network $G_{N(v)}$, we have $\tau_{G_{N(v)}}(r_1,r_2)=4$; For ego-network $G_{N(r_1)}$, we have $\tau_{G_{N(r_1)}}(v,r_2)=3$. As a result, $\tau_{G_{N(v)}}(r_1,r_2) \neq \tau_{G_{N(r_1)}}(v,r_2)$. The following observation formalizes this property of non-symmetry.

Observation 1 (Non-Symmetry). *Consider an edge $e=(v,u) \in E$ and a common neighbor $w \in N(v) \cap N(u)$. The* ego-networks *$G_{N(v)}$ and $G_{N(u)}$ have non-symmetry structure for vertex w as follows. Even if edge (u,w) in the* ego-network *$G_{N(v)}$ has $\tau_{G_{N(v)}}(u,w) \geq k$, edge (v,w) in the* ego-network *$G_{N(u)}$ may have $\tau_{G_{N(u)}}(v,w) < k$.*

In view of this result, we infer that given an edge $(v,u) \in E$, the prospects for exploiting the process of computing $score(v)$ to derive an upper bound for $score(u)$ are not promising. It shows significant challenges for deriving an upper bound. The truss-based structural diversity cannot enjoy the nice symmetry properties of component-based structural diversity [6], [19], which also brings challenges for score computation. We next investigate the structural properties of maximal connected k-truss, in search of prospects for an upper bound of $score(v)$.

An Upper Bound $\overline{score}(v)$. Consider that the smallest maximal connected k-truss is a completed graph of k vertices as k-clique. A k-clique has k vertices and $\frac{k(k-1)}{2}$ edges. Based on the analysis of ego-network size, we can infer the following useful lemma.

Lemma 2. *For a vertex $v \in V$, $score(v)$ has an upper bound of $\overline{score}(v) = \min\{\lfloor \frac{d(v)}{k} \rfloor, \lfloor \frac{2m_v}{k(k-1)} \rfloor\}$, where m_v is the number of edges in* ego-network *$G_{N(v)}$. Thus, $score(v) \leq \overline{score}(v)$ holds.*

Proof. First, $G_{N(v)}$ has $d(v)$ vertices. Since the minimum vertex size of a maximal connected k-truss is k, $G_{N(v)}$ has at most $\lfloor \frac{d(v)}{k} \rfloor$ maximal connected k-trusses in $G_{N(v)}$. Thus, $score(v) \leq \lfloor \frac{d(v)}{k} \rfloor$ holds. Second, $G_{N(v)}$ has m_v edges. Since the minimum edge size of a maximal connected k-truss is $\frac{k(k-1)}{2}$ edges, $G_{N(v)}$ has at most $\lfloor \frac{2m_v}{k(k-1)} \rfloor$ maximal connected k-trusses in $G_{N(v)}$. As a result, $score(v) \leq \min\{\lfloor \frac{d(v)}{k} \rfloor, \lfloor \frac{2m_v}{k(k-1)} \rfloor\} = \overline{score}(v)$ holds. □

Algorithm 2. Efficient Truss-Based Top-r Search Framework

```
Input: $G=(V,E)$, an integer $r$, the trussness threshold $k$
Output: Top-$r$ truss-based structural diversity results
 1: Apply the graph sparsification on $G$ by removing all edges
    $e$ with $\tau_G(e) \leq k$ and isolated nodes;
 2: for $v \in V$ do
 3:     $\overline{score}(v) \leftarrow \min\{\lfloor \frac{d(v)}{k} \rfloor, \lfloor \frac{2m_v}{k(k-1)} \rfloor\}$;
 4: $\mathcal{L} \leftarrow$ sort all vertices $V$ in descending order of $\overline{score}(v)$;
 5: $\mathcal{S} \leftarrow \emptyset$;
 6: while $\mathcal{L} \neq \emptyset$
 7:     $v^* \leftarrow \arg\max_{v \in \mathcal{L}} \overline{score}(v)$; Delete $v^*$ from $\mathcal{L}$;
 8:     if $|\mathcal{S}| = r$ and $\overline{score}(v^*) \leq \min_{v \in \mathcal{S}} score(v)$ then
 9:         break;
10:     Computing $score(v^*)$ using Algorithm 1;
11:     if $|\mathcal{S}| < r$ then $\mathcal{S} \leftarrow \mathcal{S} \cup \{v^*\}$;
12:     else if $score(v^*) > \min_{v \in \mathcal{S}} score(v)$ then
13:         $u \leftarrow \arg\min_{v \in \mathcal{S}} score(v)$;
14:         $\mathcal{S} \leftarrow (\mathcal{S} - \{u\}) \cup \{v^*\}$;
15: return $\mathcal{S}$ and their social contexts $\mathsf{SC}(v)$ for $v \in \mathcal{S}$;
```

4.3 An Efficient Top-r Search Framework

Equipped with graph sparsification and an upper bound $\overline{score}(v)$, we propose our efficient truss-based top-r search framework as follows.

Algorithm. Algorithm 2 outlines the details of truss-based top-r search framework. It first performs graph sparsification by applying truss decomposition on graph G and removing all the edges e with $\tau_G(e) \leq k$ and isolated nodes from G (line 1). Then, it computes the upper bound of $\overline{score}(v)$ for each vertex $v \in V$ and sorts them in the decreasing order in $\mathcal{L}$ (lines 2-4). Next, the algorithm iteratively pops out a vertex v^* with the largest $\overline{score}(v)$ from $\mathcal{L}$ (line 7). After that, the algorithm checks an early stop condition. If the answer set $\mathcal{S}$ has r vertices and $\overline{score}(v^*) \leq \min_{v\in\mathcal{S}} score(v)$ holds, we can safely prune the remaining vertices in $\mathcal{L}$ and early terminate (lines 8-9); otherwise, it needs to invoke Algorithm 1 to compute structural diversity $score(v^*)$ (line 10) and checks whether v^* should be added into the answer set $\mathcal{S}$ (lines 11-14). Finally, it outputs the top-r results $\mathcal{S}$ and their social contexts $\mathsf{SC}(v)$ for $v \in \mathcal{S}$ (line 15).

Example 2. We apply Algorithm 2 on graph G in Fig. 1. Assume that $k = 4$ and $r = 1$. $\mathcal{L}$ ranks all vertices in the decreasing order of their upper bounds. At the first iteration, the vertex v in G has the highest upper bound $\overline{score}(v) = 3$ of $\mathcal{L}$. It then computes $score(v) = 3$ and adds v into the answer set $\mathcal{S}$. At the next iteration, the highest upper bound of vertices in $\mathcal{L}$ is 1 (e.g., $\overline{score}(x_1) = 1$), which triggers the early termination (lines 8-9 of Algorithm 2). That is, $|\mathcal{S}| = 1$ and $\overline{score}(v^*) = 1 \leq \min_{v\in\mathcal{S}} score(v) = 3$. The algorithm terminates with an answer $\mathcal{S} = \{v\}$. During the whole computing process, it invokes Algorithm 1 only once for structural diversity calculation, which is much less than 17 times by the online search algorithm in Section 3. It demonstrates the pruning power of top-r search framework.

4.4 Complexity Analysis

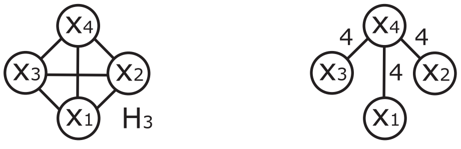
We analyze the complexity of Algorithm 2. Let the reduced graph be $G' \subseteq G$. Let ρ', m', and $\mathcal{T}'$ be respectively the arboricity, the number of edges, and the number of triangles in G'. Obviously, $\rho' \leq \rho$, $m' \leq m$, and $\mathcal{T}' \leq \mathcal{T}$. First, graph sparsification takes $O(\rho m)$ time by truss decomposition for graph G. Second, computing the upper bounds for all vertices takes $O(\rho' m')$ time on the reduced graph G'. In addition, $\mathcal{L}$ performs vertex sorting in the order of $\overline{score}(v^*)$ and maintains the list, which can be done in $O(n)$ time. In the worst case, Algorithm 2 needs to compute $score(v)$ for every vertex v, which takes $O(\rho'(m' + \mathcal{T}'))$ by Theorem 2. Overall, Algorithm 2 takes $O(\rho'(m' + \mathcal{T}') + \rho m + n) \subseteq O(\rho m + \rho' \mathcal{T}')$ time and $O(m)$ space.

5 A Novel Index-Based Approach

Algorithm 2 is still not efficient for large networks, because the operation of computing $score(v)$ in Algorithm 1 applies truss decomposition on each ego-network $G_{N(v)}$ from scratch in an online manner, which is highly expensive. It wastes lots of computations on the unnecessary access of disqualified edges whose trussnesses are less than k in the ego-network. To further speed up the calculation of $score(v)$, in this section, we develop a novel truss-based structural diversity index (TSD-index). TSD-index is a compact and elegant tree structure to keep the structural diversity information for all ego-networks in G. Based on TSD-index, we design a fast solution of computing $score(v)$ and propose an index-based top-r search approach to quickly find r vertices with the highest scores, which is particularly efficient to handle multiple queries with different r and k on the same graph G.

5.1 TSD-Index Construction

An intuitive indexing approach is to keep all maximal connected k-trusses in $G_{N(v)}$ by storing the trussness for all edges. However, it requires $O(\mathcal{T})$ space to store all ego-networks $G_{N(v)}$ for each vertex $v \in V$, which is inefficient for large networks. To develop efficient indexing scheme, we first start with the following observations.



(a) H_3 (b) Tree representation of H_3

Fig. 3. An example of Observation 2.

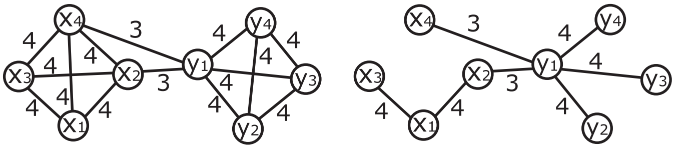
Observation 2. *Fig. 3a depicts a maximal connected 4-truss H_3 in the* ego-network $G_{N(v)}$ *in Fig. 1b. The definition of truss-based structural diversity only focuses on the number of maximal connected k-trusses, but ignores the connections between vertices in a maximal connected k-truss. It indicates that we do not need to store its whole structure. Fig. 3b shows a tree-shaped structure with edge weights, which can clearly represent that x_1, x_2, x_3, x_4 are in the same maximal connected 4-truss.*

Observation 3. *Fig. 4a depicts a maximal connected 3-truss H_1 in the* ego-network $G_{N(v)}$ *in Fig. 1b. A tree structure is enough to represent the connectivity of vertices. However, if we keep an arbitrary tree structure of H_1 to connect all vertices, information loss of maximal connected k-trusses may happen. Consider the tree in Fig. 4b, for vertex x_4, it has no edges connecting with x_1, x_2 and x_3, but one incident edge with a weight of 3. From this tree structure in Fig. 4b, we cannot infer that x_4 is involved in a maximal connected 4-truss H_3 shown in Fig. 3a.*

In summary, Observation 2 shows that the tree-shaped structure is enough to represent the identity of a maximal connected k-truss. Observation 3 further shows that the tree-shaped structure should have the maximum edge trussnesses to ensure no loss information of structural diversity, indicating a maximum spanning forest of $G_{N(v)}$ with the largest total weights of edge trussness.

TSD-Index Structure. Based on the above observations, we are able to design our index structure of TSD-index. We first define a weighted graph WG_v for a vertex $v \in V$. WG_v has the same vertex set and edge set with $G_{N(v)}$ and $\forall e \in E(WG_v)$ has a weight $w(e) = \tau_{G_{N(v)}}(e)$. In other words, we assign a weight on each edge with its trussness on ego-network $G_{N(v)}$ to form WG_v. As a result, the TSD-index of $G_{N(v)}$ is defined as the maximum spanning forest of WG_v, denoted by TSD_v.

TSD-Index Construction. Algorithm 3 describes a method of TSD-index construction on graph G. The algorithm constructs the TSD-index for each vertex $v \in G$ (lines 1-10). It first performs truss decomposition on $G_{N(v)}$ to obtain all edge trussnesses (line 2). The algorithm then constructs a weighted graph WG_v for $G_{N(v)}$ where each edge e has a weight $w(e) = \tau_{G_{N(v)}}(e)$ (line 3). Let TSD_v be initially as all isolated vertices $N(v)$ (line 4). Then, we construct the maximum spanning forest of WG_v by adding edges in the decreasing order of edge weights one by one into TSD_v (lines 5-10). Let $\mathcal{L}$ be the edge set of WG_v $E(WG_v)$. We visit each edge $e = (u, w)$ in the decreasing order of weight $w(e)$ in $\mathcal{L}$, and check whether u, w are in the same component in TSD_v. If u, w are disconnected, we add an edge



(a) H_1 (b) Inaccurate tree

Fig. 4. An example of Observation 3.

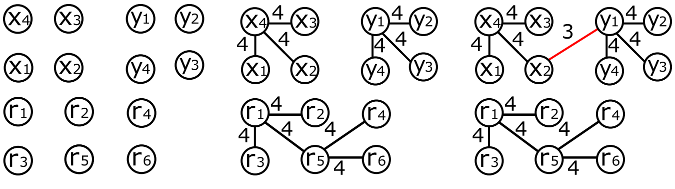
connecting u and w in TSD_v. The process of constructing TSD_v breaks when all edges have been visited in $\mathcal{L}$ (lines 6-10). Algorithm 3 returns the TSD-index of G as $\{\mathsf{TSD}_v | v \in V\}$.

Example 3. Fig. 5 illustrates the TSD-Index construction of TSD_v for a vertex v in graph G in Fig. 1. Fig. 5a shows that TSD_v is initialized to be a set of isolated nodes $N(v)$. Then, it checks all 4-truss edges and add qualified edges one by one into TSD_v. According to Observation 2, when Algorithm 3 processes the edge (x_3, x_1), it finds that x_3 and x_1 are in the same component in Fig. 5a, thus (x_3, x_1) is not added to TSD_v in Fig. 5b. Afterwards, it adds the edge $e = (x_2, y_1)$ with weight $w(e) = 3$ into TSD_v in Fig. 5c. The complete structure of TSD_v is finally depicted in Fig. 5c.

Algorithm 3. TSD-Index Construction

```
Input: $G = (V, E)$
Output: TSD-index of $G$
1: for $v \in V$ do
2:     Apply the truss decomposition on $G_{N(v)}$;
3:     Construct a weighted graph $WG_v$ for $G_{N(v)}$, where each
       edge $e$ in $WG_v$ has a weight $w(e) = \tau_{G_{N(v)}}(e)$;
4:     Let a forest $\mathsf{TSD}_v$ formed by all isolated vertices $N(v)$;
5:     Let an edge set $\mathcal{L} \leftarrow E(WG_v)$;
6:     while ($\mathcal{L} \neq \emptyset$)
7:         Let $e = (u, w) \in \mathcal{L}$ has the largest weight $w(e)$ in $\mathcal{L}$;
8:         if vertices $u$ and $w$ are disconnected in $\mathsf{TSD}_v$ then
9:             Add a new edge $e$ with its weight $w(e)$ into $\mathsf{TSD}_v$;
10:        Delete $e$ from $\mathcal{L}$;
11: return $\{\mathsf{TSD}_v | v \in V\}$;
```

Remarks. Note that our TSD-index can answer queries of any k and r. It is independent to parameters k and r once the TSD-index is constructed. TSD-index can not only be used for calculating the structural diversity scores, but also support the retrieval of all social contexts in ego-networks. Early pruning (Property 1 and Lemma 2) works for the online search algorithms, but not for TSD-index construction in Algorithm 3.



(a) Step-1: Initialization with $N(v)$. (b) Step-2: adding 4-truss edges. (c) Step-3: adding 3-truss edges.

Fig. 5. Illustration of TSD-Index construction of TSD_v.

5.2 TSD-Index-Based Top-r Search

In the following, we first propose an efficient algorithm for computing structural diversity scores using the TSD-index. Based on it, we develop our TSD-index-based top-r search algorithm.

Computing $score(v)$ Based on TSD-Index. Algorithm 4 presents a method of computing $score(v)$ based on the TSD-index. The algorithm first retrieves a subgraph H of TSD_v formed by all edges e with the edge weight $w(e) \geq k$ (line 1). Next, it finds all maximal connected k-trusses of H that are the social contexts $\mathsf{SC}(v)$ (lines 2-6). Applying the breadth-first-search strategy, it uses one hashtable to ensure each vertex to be visited once, and one queue to visit the vertices of a neighborhood social context S one by one (lines 3-6). After traversing each component in H, it keeps the social context $\mathsf{SC}(v)$ by the union of S (line 6). Finally, it returns $score(v)$ as the multiplicity of social contexts $\mathsf{SC}(v)$ (lines 7-8).

Algorithm 4. Computing $score(v)$ Based on TSD-index

```
Input: $G = (V, E)$, a vertex $v$, the trussness threshold $k$
Output: $score(v)$
1: Let $H$ be a subgraph of $\mathsf{TSD}_v$ formed by all edges $e$ with
   $w(e) \geq k$;
2: $\mathsf{SC}(v) \leftarrow \emptyset$;
3: for each unvisited vertex $u \in V(H)$ do
4:     Traverse the component $X$ containing $u$ in $H$;
5:     Let a social context $S \leftarrow$ the set of vertices in $X$;
6:     $\mathsf{SC}(v) \leftarrow \mathsf{SC}(v) \cup \{S\}$;
7: score$(v) \leftarrow |\mathsf{SC}(v)|$;
8: return score$(v)$;
```

TSD-index-*Based Top-r Search Algorithm.* Based on the TSD_v, we design a new upper bound of $score(v)$ for pruning. The upper bound of $score(v)$ is defined as $\widetilde{score}(v) = \frac{|\{e \in \mathsf{TSD}_v : w(e) \geq k\}|}{k-1}$. The essence of $\widetilde{score}(v)$ holds because a maximal connected k-truss should have a tree-shaped representation of at least $(k-1)$ edges with weights of no less than k in TSD_v. We can make a fast calculation of $\widetilde{score}(v)$ by sorting all edges of TSD_v in the decreasing order of edge weights, during the index construction. Equipped with Algorithm 4 of computing $score(v)$ and a new upper bound $\widetilde{score}(v)$, our TSD-index-based top-r structural diversity search algorithm invokes an efficient framework similarly as Algorithm 2, which finds the top-r answers by pruning those vertices v that has the upper bound $\widetilde{score}(v)$ no greater than the top-r answer $\mathcal{S}$.

5.3 Complexity Analysis

Theorem 3. *Algorithm 3 constructs* TSD-index *for a graph G in $O(\rho(m + \mathcal{T}))$ time and $O(m)$ space. The index size is $O(m)$. Moreover,* TSD-index*-based search approach tackles the problem of truss-based structural diversity search in $O(m)$ time and $O(m)$ space.*

Proof. First, we analyze the time complexity of TSD construction. For each vertex $v \in V$, Algorithm 3 extracts $G_{N(v)}$ and applies truss decomposition on $G_{N(v)}$. This totally takes $O(\rho(m + \mathcal{T}))$ by Theorem 2. In addition, for $v \in V$, a weighted graph WG_v has n_v vertices and m_v

edges. The sorting of weighted edges can be done in $O(m_v)$ time using a bin sort. Thus, applying Kruskal's algorithm [10] to find the maximum spanning forest from WG_v takes $O(m_v)$ time. As a result, constructing the TSD-index for all vertices takes $O(\sum_{v\in V} m_v) \subseteq O(\mathcal{T})$. Therefore, the time complexity of Algorithm 3 is $O(\rho(m+\mathcal{T}))$ in total.

Second, we analyze the space complexity of TSD construction. The edge set $\mathcal{L}$ takes $O(m_v) \subseteq O(m)$ space. The index TSD_v takes $O(n_v) \subseteq O(n)$ space. The space complexity of Algorithm 3 is $O(m+n) \subseteq O(m)$.

Third, we analyze the index size of TSD-index of G. For a vertex v, TSD_v is the maximum spanning forest of WG_v, which has no greater than $n_v - 1$ edges. Thus, the size of TSD_v is $O(n_v)$. Overall, the index size of TSD-index of G is $O(\sum_{v\in V} n_v) \subseteq O(m)$.

Finally, we analyze the time and space complexity of TSD-index-based search approach. First, Algorithm 4 takes $O(|N(v)|)$ time to compute $score(v)$ for a vertex $v \in V$. In the worst case, the TSD-index-based search approach needs to invoke Algorithm 4 to compute $score(v)$ for all vertices. It takes $O(\sum_{v\in V} |N(v)|) \subseteq O(m)$ time complexity. In addition, the upper bound $\widetilde{score}(v)$ takes $O(1)$ space for each vertex $v \in V$. Thus, the space complexity is $O(m)$. □

Remarks. In summary, the TSD-index-based search approach is clearly faster than the online search algorithms and Algorithm 2, in terms of their time complexities. In addition, TSD-index can support efficient updates in dynamic graphs where the graph structure undergo frequently updates with nodes/edges insertions/deletions. Although an edge insertion may cause the structure change of many ego-networks, the updating techniques are still promising to be further developed with some carefully designed ideas, given by the existing theory and algorithms of k-truss updating on dynamic graphs [20], [38].

6 A Global GCT-Index-Based Approach

In this section, we propose a new approach GCT for truss-based structural diversity search, which utilizes the global triangle information for efficient ego-network truss decomposition and develops a compressed truss-based diversity GCT-index to improve TSD-index.

6.1 Solution Overview

We briefly introduce a solution overview of GCT algorithm, which leverages one-shot global triangle listing and a compressed GCT-index for fast structural diversity search computation. The method of GCT-index construction is outlined in Algorithm 5. GCT-index equips with three new techniques and implementations: 1) fast ego-network extraction (lines 1-4 of Algorithm 5); 2) bitmap-based truss decomposition (lines 5-14 of Algorithm 5); and 3) GCT-index construction for an ego-network (line 15 of Algorithm 5), which is detailed presented in Algorithm 6.

Note that it is very useful but non-trivial challenging to explore the sharing computation across vertices using global truss decomposition. We observe that the one-shot triangle listing of global truss decomposition can help to efficiently extract ego-networks for all vertices. Moreover, we realize that the bitwise operations can further improve the efficiency of truss decomposition in such local ego-networks. In addition, we propose a compact index structure of GCT-index, which maintains only supernodes and superedges to discard the edges within the same k-level of social contexts. GCT-index based query processing can be done more efficient than the TSD-index-based approach.

Algorithm 5. GCT-index Construction

```
Input: Graph $G$
Output: GCT-index of all vertices
1:  Let $G_{N(v)}$ be an empty graph for each $v \in V$;
2:  for each edge $e = (u, v) \in E$ do
3:      for each vertex $w \in N(u) \cap N(v)$ do
4:          Add the new edge $e$ into $G_{N(w)}$;
5:  for each vertex $v$ in $G$ do
6:      Retrieve an ego-network $G_{N(v)}$ directly based on Steps 2-4, which avoids the duplicate triangle listing;
7:      Give IDs to all vertices in $G_{N(v)}$ sequentially from 1 to $L$, where $L = |N(v)|$.
8:      for each vertex $u \in N(v)$ do
9:          Create a bitmap $\mathsf{Bits}_u$ of all 0 bits with $|\mathsf{Bits}_u| = L$.
10:         for each vertex $w \in N_{G_{N(v)}}(u)$ do
11:             $\mathsf{Bits}_u[w] \leftarrow 1$;
12:     for each edge $e = (u, w) \in E(G_{N(v)})$ do
13:         $\sup_{G_{N(v)}}(e) \leftarrow \mathsf{Bits}_x$ AND $\mathsf{Bits}_y$;
14:     Apply a bitmap-based peeling process for truss decomposition [36] on $G_{N(v)}$;
15:     Apply GCT-index construction in Algorithm 6 on $G_{N(v)}$ to obtain $\mathsf{GCT}_v$;
16: return the GCT-index $\{\mathsf{GCT}_v : v \in V\}$;
```

6.2 Fast Ego-Network Truss Decomposition

In this section, we propose a fast method of ego-network truss decomposition, which leverages on the global triangle listing and bitmap-based truss decomposition.

Global Triangle Listing based Ego-Network Extraction. Ego-network extraction is the first key step of score computation in Algorithm 1 and TSD-index construction in Algorithm 3. However, it suffers from heavily duplicate triangle listing. Specifically, for each vertex v, it needs to perform a triangle listing to find all triangles $\triangle_{vuw}$ and generate an edge (u, w) in ego-network $G_{N(v)}$. $\triangle_{vuw}$ is generated twice, which checks the common neighbors of $N(v) \cap N(u)$ and $N(v) \cap N(w)$ for two edges (v, u) and (v, w) respectively. Similarly, for vertices u and w, $\triangle_{vuw}$ is generated twice respectively for extracting ego-networks $G_{N(u)}$ and $G_{N(w)}$. Unfortunately, $\triangle_{vuw}$ is repeatedly enumerated for six times, which is inefficient for local ego-network extraction.

To this end, we propose to utilize global triangle listing once to generate all the ego-networks in G. The details of fast ego-network extraction is presented in Algorithm 5 (lines 1-4). Specifically, for each edge $e = (u, v) \in E$, it identifies triangle $\triangle_{vuw}$ by enumerating all the common neighbors $w \in N(u) \cap N(v)$, and adds edge e into ego-network $G_{N(w)}$ (lines 2-4). Thus, it finishes the construction for all ego-networks, which can be directly used in the following ego-network truss decomposition. Each triangle $\triangle_{vuw}$ is enumerated for three times, which saves a half of original computations using six enumeration times. Overall, our method of fast ego-network extraction makes use of global triangle listing for best sharing in local ego-network computations.

Bitmap-Based Truss Decomposition. We propose a bitmap-based approach to accelerate the truss decomposition. To apply truss decomposition on an obtained ego-network $G_{N(v)}$, an important step is support computation, i.e., calculating $\sup_{G_{N(v)}}(e)$ as the number of triangles containing $e = (x, y)$ for each edge $e \in E(G_{N(v)})$. The existing method of computing $\sup_{G_{N(v)}}(e)$ [36] uses the triangle listing, which checks each neighbor $z \in N(x)$ in ego-network $G_{N(v)}$ to see whether $z \in N(y)$ using hashing technique. The hash checking takes constant time $O(1)$ in theoretical analysis, but in practice costs an expensive time overhead of support computation appeared in large graphs for frequent hash updates and checks. To this end, we propose to use a bitmap technique to accelerate the support computation. First, we give an order ID to every vertex in $G_{N(v)}$ sequentially from 1 to L, where $L = |N(v)|$. For each vertex $x \in N(v)$, we create a binary bitmap Bits_x with all 0 bits. For each edge $e = (x, y) \in E(G_{N(v)})$, we set to 1 for both the xth bit of bitmap Bits_y and the yth bit of bitmap Bits_x, indicating $x \in N_{G_{N(v)}}(y)$ and $y \in N_{G_{N(v)}}(x)$. Then, the support of $\sup(e)$ equals to the number of 1 bits commonly appeared in Bits_x and Bits_y, denoted by $\sup_{G_{N(v)}}(e) = |N(u) \cap N(v)| = \mathsf{Bits}_x$ AND Bits_y. Note that the binary operation of bitwise AND can be done efficiently.

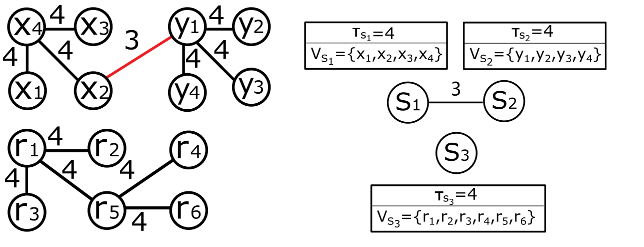
Algorithm 5 presents the detailed procedure of bitmap-based truss decomposition (lines 5-15). The algorithm first retrieves ego-network $G_{N(v)}$ directly from the global triangle listing (line 6). It then initializes the Bits_x for all vertices $x \in N(v)$ and calculates the support $\sup_{G_{N(v)}}(e)$ as Bits_x AND Bits_y for all edges $e \in E(G_{N(v)})$ (lines 8-13). Next, The algorithm applies a bitmap-based peeling process for truss decomposition [36] on $G_{N(v)}$. Specifically, when an edge (x, y) is removed from a graph, it updates $\mathsf{Bits}_x[y] = 0$ and $\mathsf{Bits}_y[x] = 0$. Due to the limited space, we omit the details of similar bitmap-based peeling process (line 14). After obtaining all the edge trussnesses, we invoke Algorithm 6 (to be introduced in Section 6.3) to construct GCT-index (line 15).

6.3 GCT-Index Construction and Query Processing

In this section, we propose a new data structure of GCT-index, which compresses the structure of TSD-index in a more compact way.

We start with discussing the limitations of TSD-index. Each social context is defined as a maximal connected k-truss. The spanning forest structure of TSD-index stores not only the edge connections between different social contexts, but also the internal edges within a social context. However, such information of internal edges is redundant, which can be avoided for indexing. For example, consider the TSD-index of vertex v in Fig. 6a. The vertices $\{x_1, x_2, x_3, x_4\}$ form a social context of maximal connected 4-truss. The edges (x_4, x_1), (x_4, x_2), and (x_4, x_3) can be ignored for indexing storage. Instead, we keep a node list of $\{x_1, x_2, x_3, x_4\}$, which is enough to recover the information of social contexts by saving the cost of edge listing.

GCT-Index Structure. GCT-index keeps a maximum-weight forest-like structure similar as TSD-index, which consists of supernodes and superedges. Specifically, for a vertex v, the GCT-index of v is denoted by $\mathsf{GCT}_v = (\mathcal{V}_v, \mathcal{E}_v)$, where $\mathcal{V}_v$ and $\mathcal{E}_v$ are the set of supernodes and superedges respectively. A supernode $S \in \mathcal{V}_v$ represents a group of vertices that are connected via the edges of the same trussness $\tau(S_u)$ in a social context. Each supernode is associated with two features, including the trussness of connecting edges $\tau(S_u)$ and a vertex list V_S of the vertices belonging to this social context. Based on the isolated supernodes of $\mathcal{V}_v$, we add the superedges $\mathcal{E}_v = \{(S_i, S_j) : S_i, S_j \in \mathcal{V}_v \text{ and } \exists v_i \in V_{S_i}, v_j \in V_{S_j} \text{ such that the edge } (v_i, v_j) \in E\}$ into GCT_v, such that all vertices form a forest with the largest weight. Note that the weight of a superedge $(S_i, S_j) \in \mathcal{E}_v$ is denoted by the corresponding edge trussness in $G_{N(v)}$, i.e., $w((S_i, S_j)) = \max_{v_i \in V_{S_i}, v_j \in V_{S_j}} \tau_{G_{N(v)}}(v_i, v_j)$. For example, for a vertex v, the corresponding TSD-index in Fig. 6a is compressed into a small GCT-index GCT_v as shown in Fig. 6b. $\mathsf{GCT}_v = (\mathcal{V}_v, \mathcal{E}_v)$ where $\mathcal{V}_v = \{S_1, S_2, S_3\}$ and $\mathcal{E}_v = \{(S_1, S_2)\}$. The supernode S_1 consists of $\tau(S_1) = 4$ and $V_{S_1} = \{x_1, x_2, x_3, x_4\}$ that belong to 4-truss social context. The superedge (S_1, S_2) has a weight of $w((S_1, S_2)) = 3$, due to $\tau_{G_{N(v)}}((x_2, y_1)) = 3$. This edge indicates that the vertices in S_1 and S_2 belong to the same 3-truss social context, i.e., $V_{S_1} \cup V_{S_2} = \{x_1, x_2, x_3, x_4, y_1, y_2, y_3, y_4\}$.



(a) TSD-index TSD_v (b) GCT-index GCT_v

Fig. 6. GCT_v is a compressed data structure of TSD_v for vertex v in graph G as shown in Fig. 1a.

Algorithm 6. GCT-index Construction for an Ego-Network

```
Input: an ego-network $G_{N(v)}$ for a vertex $v$
Output: GCT-index of $v$
 1: $\mathcal{V}_v \leftarrow \emptyset$; $\mathcal{E}_v \leftarrow \emptyset$;
 2: for each vertex $u \in N(v)$ do
 3:   Super-node $S_u$: $\tau(S_u) = \tau_{G_{N(v)}}(u)$ and $V_{S_u} = \{u\}$;
 4:   $\mathcal{V}_v \leftarrow \mathcal{V}_v \cup \{S_u\}$;
 5: Let an edge set $\mathcal{L} \leftarrow E(G_{N(v)})$;
 6: while $\mathcal{L} \neq \emptyset$ do
 7:   Pop out an edge $e = (u, w) \in \mathcal{L}$ with the largest trussness $\tau_{G_{N(v)}}(e)$ from $\mathcal{L}$;
 8:   Identify the corresponding supernodes $S_u$ and $S_w$ for $u$ and $w$ respectively.
 9:   if $S_u = S_w$ or $S_u$ and $S_w$ are connected then continue;
10:   if $\tau(S_u) = \tau(S_w) = \tau_{G_{N(v)}}(e)$ then
11:     Two supernodes merge: $V_{S_u} \leftarrow V_{S_u} \cup V_{S_w}$;
12:     Assign all $S_w$'s incident edges to $S_u$ and delete $S_w$;
13:   else
14:     Superedge insertion: $\mathcal{E}_v \leftarrow \mathcal{E}_v \cup (S_u, S_w)$;
15:     $w((S_u, S_w)) \leftarrow \tau_{G_{N(v)}}(e)$;
16: return $\mathsf{GCT}_v = (\mathcal{V}_v, \mathcal{E}_v)$;
```

GCT-Index Construction. Algorithm 6 presents the procedures of constructing GCT-index in an ego-network $G_{N(v)}$ for a vertex v. The algorithm first creates the supernodes S_u

for each vertex u in ego-network $G_{N(v)}$ (lines 2-4). For each supernode S_u, the trusssness $\tau(S_u)$ is initialized as the vertex trussness of $\tau_{G_{N(v)}}(u)$ and $V_{S_u} = \{u\}$ (line 3). Next, the algorithm continues to construct GCT-index by adding superedges and merging supernodes, via a traverse of the whole set of edges $L = E(G_{N(v)})$ (lines 5-15). In each iteration, it retrieves an edge $e = (u, w)$ with the largest trussness in L (line 7). If two vertices u and w belong to the same supernode or their supernodes S_u and S_w are already connected in GCT_v, then it continues to check the next edge in L (lines 8-9). If two different supernodes S_u and S_w have the same trussnesses as $\tau_{G_{N(v)}}(e)$, it merges two supernodes into one by assigning all S_w's feature to S_u. Specifically, it unites two vertex lists as $V_{S_u} = V_{S_u} \cup V_{S_w}$ and assigns to S_u the edges that are incident to supernode S_w, and then removes S_w from $\mathcal{V}_v$ (lines 10-12); Otherwise, it adds a superedge between S_u and S_w and assigns the edge weight as $w((S_u, S_w)) = \tau_{G_{N(v)}}(e)$ (lines 14-15). After processing all edges in L, the algorithm finally returns the GCT-index as $\mathsf{GCT}_v = (\mathcal{V}_v, \mathcal{E}_v)$ (line 16).

GCT-index*-Based Query Processing*. Thanks to a very elegant and compact structure of GCT-index, we next introduce a fast method to compute $score(v)$ for a given vertex v.

Lemma 3. *For a vertex $v \in V$ and a number k, the structural diversity score of v is $score(v) = N_k - M_k$, where N_k and M_k are the number of supernodes and superedges with trussness no less than k in GCT_v, i.e., $N_k = |\{S \in \mathcal{V}_v : \tau(S) \geq k\}|$ and $M_k = |\{e \in \mathcal{E}_v : \tau(e) \geq k\}|$.*

Proof. Let $score(v) = x$ w.r.t. a particular k. This indicates that ego-network $G_{N(v)}$ has x social contexts. In terms of the structural properties of GCT-index, each maximal connected k-truss is represented by a connected structure of spanning tree or just one single supernode. In the ith spanning tree (or ith single supernode), the number of supernodes is denoted as n_i, and the number of superedges is $n_i - 1$. Thus, $N_k = \sum_{i=1}^{x} n_i$ and $M_k = \sum_{i=1}^{x} n_i - 1$. As a result, $N_k - M_k = \sum_{i=1}^{x} n_i - \sum_{i=1}^{x}(n_i - 1) = \sum_{i=1}^{x} 1 = x$.

Note that the GCT-index-based query processing for structural diversity search takes $O(m)$ time in worst, where m is the number of edges in G. □

7 Experiments

In this section, we evaluate the effectiveness and efficiency of our proposed algorithms on real-world networks. All algorithms mentioned above are implemented in C++ and complied by gcc at -O3 optimization level.

Datasets. We use eight datasets of real-world networks, and treat them as undirected graphs. Except for socfb-konect,[1] all other datasets are available from the Stanford Network Analysis Project [26]. The network statistics are described in Table 1. We report the node size $|V|$, the edge size $|E|$, the maximum degree d_{max}, the maximum edge trussness $\tau_G^* = \max_{e \in E} \tau_G(e)$, the maximum edge trussness among all ego-networks $\tau_{ego}^* = \max_{v \in V, e \in E(G_{N(v)})} \{\tau_{G_{N(v)}}(e)\}$, and the number of triangles $\mathcal{T}$.

TABLE 1
Network Statistics(K$= 10^3$ and M$= 10^6$)

Name	$\|V\|$	$\|E\|$	d_{max}	τ_G^*	τ_{ego}^*	$\mathcal{T}$
Wiki-Vote	7**K**	103**K**	1,065	23	22	608,389
Email-Enron	36**K**	183**K**	1,383	22	21	727,044
Epinions	75**K**	508**K**	3,044	33	32	1,624,481
Gowalla	196**K**	950**K**	14,730	29	28	2,273,138
NotreDame	325**K**	1.4**M**	10,721	155	154	8,910,005
LiveJournal	4**M**	34.7**M**	14,815	352	351	177,820,130
socfb-konect	59**M**	92.5**M**	4,960	7	6	6,378,280
Orkut	3.1**M**	117**M**	33,313	73	72	412,002,900

Compared Methods and Evaluated Metrics. To evaluate the effectiveness of top-r truss-based structural diversity model, we conduct the simulation of social influence process and report the number of affected vertices of the r selected vertices by all methods. We test and compare our truss-based structural diversity method with three other methods as follows.

- Random: is to select r vertices from graph by random.
- Comp-Div: is to select r vertices with the highest k-sized component-based structural diversity [6].
- Core-Div: is to select r vertices with the highest k-core-based structural diversity [18].
- Truss-Div: is our method by selecting r vertices with the highest k-truss-based structural diversity.

In addition, to evaluate the efficiency of improved strategies, we compare our algorithms with two state-of-the-art methods Comp-Div [6] and Core-Div [18]. Note that the implementation of Comp-Div in [6] is much faster than the method in [19]. We also test and compare four algorithms proposed in this paper as follows.

- baseline: is the online search algorithm that computes structural diversity for all vertices in Section 3.
- bound: is the efficient approach using graph sparsification and an upper bound for pruning vertices in Algorithm 2.
- TSD: is the TSD-index based approach, which uses Algorithm 4 to compute structural diversity.
- GCT: is the GCT-index based approach in Algorithm 5.

We compare them by reporting the running time in seconds and the search space as the number of vertices whose structural diversities are computed in search process. The less running time and search space are, the better efficiency performance is.

Parameters. We set the parameters $r = 100$ and $k = 3$ by default. We also evaluate the methods by varying the parameters k in {2, 3, 4, 5, 6} and r in $\{50, 100, 150, 200, 250, 300\}$.

7.1 Efficiency Evaluation

Exp-1 (Efficiency Comparison on All Datasets). We compare the efficiency of our proposed methods on all datasets. Table 2 shows the results of running time and search space. Clearly, TSD is the most efficient in terms of running time, and baseline is the worst. TSD uses less search space than bound, indicating a stronger pruning ability of $\widetilde{score}(v)$ against $\overline{score}(v)$ in Lemma 2. The speedup ratio R_t between TSD and baseline is defined by $R_t = t_{\mathsf{baseline}}/t_{\mathsf{TSD}}$ where t_{baseline} and t_{TSD} are the running time of baseline and TSD

[1] http://networkrepository.com/socfb_konect.php

TABLE 2
Comparison of Running Time (in Seconds) and Search Space (the Number of Vertices Whose Structural Diversity are Computed) of Different Algorithms

Network	Running Time				Search Space			
	baseline	bound	TSD	R_t	baseline	bound	TSD	R_s
Wiki-Vote	10.7s	10.2s	7.0ms	1,529	8,297	2,704	2,628	3.1
Email-Enron	11.8s	11.3s	18.2ms	648	36,692	4,284	4,274	8.6
Epinions	37.7s	34.2s	31.9ms	1,182	75,887	6,810	6,531	11.6
Gowalla	52.2s	42.2s	70.2ms	743	196,591	22,267	21,674	9.0
NotreDame	291s	283s	106ms	2,745	325,729	24,285	24,188	13.4
LiveJournal	10,418s	9,456s	4.9s	2,126	4,036,537	208,722	182,646	22.1
socfb-konect	1,591s	15.3s	6s	265	59,216,214	18,630	17,649	3,355
orkut	21,381s	18,071s	10.7s	1,998	3,072,626	370,343	353,606	8.6

Here $k = 3$ and $r = 100$.

respectively. The speedup ratio R_t (column 5 in Table 2) ranges from 265 to 2,745. In other words, our method TSD achieves up to 2,745X speedup on the network NotreDame. In addition, the pruning ratio R_s between TSD and baseline is defined by $R_s = S_{\textsf{baseline}}/S_{\textsf{TSD}}$ where $S_{\textsf{baseline}}$ and $S_{\textsf{TSD}}$ are the search space of baseline and TSD respectively. The pruning ratio R_s (column 9 in Table 2) ranges from 3.1 to 3,355. The pruning performance of bound and TSD are close and the best as the two upper bounds are derived in a similar way. However, as an index-based method, TSD is much faster than bound.

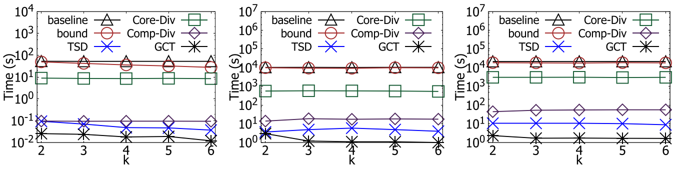
Exp-2 (Efficiency Comparison of All Different Methods). We vary parameter k to compare the efficiency of all different methods. We compare six methods of baseline, bound, TSD, GCT, Comp-Div, and Core-Div on three datasets Gowalla, Livejournal, and Orkut. The results of running time and search space are respectively reported in Figs. 7 and 8. Similar results can be also observed on other datasets. GCT is a clear winner for the varied k on all datasets. Thanks to efficient GCT-index, GCT significantly outperforms two state-of-the-art methods of Comp-Div and Core-Div on large networks of LiveJournal and Orkut. Moreover, GCT outperforms TSD, indicating the superiority of a more compact GCT-index against TSD-index. In addition, we report the search space results in Fig. 8. It shows that the search space is significantly reduced by bound against baseline on all datasets, indicating the technical superiority of graph sparsification and the upper bound of $\overline{score}(v)$. TSD performs the best in search space by leveraging another tight upper bound $\widetilde{score}(v)$, which learns structural information from the TSD-index.

Exp-3 (Indexing Scheme Comparison Between TSD *and* GCT*)*. We compare two indexing methods of TSD and GCT in terms of index construction time, index size, and index-based query processing time of structural diversity search. The results of TSD and GCT on all dataset are reported in Table 3. First, the index size of GCT-index is smaller than the size of TSD, due to a compact structure of GCT-index by discarding unnecessary edges within social contexts. Second, GCT achieves a much faster index construction time than TSD, thanks to the efficient techniques of fast ego-network extraction and bitmap-based truss decomposition. More specifically, Table 4 reports the detailed running time of ego-network extraction and ego-network truss decomposition by TSD and GCT on all datasets. The results reflect that GCT achieves significant accelerations on both ego-network extraction and ego-network truss decomposition, which validates the superiority of our speed up techniques proposed in Section 6. Finally, as shown in the columns 7 and 8 of Table 3, GCT runs much faster than TSD in terms of the query time of structural diversity search.

Exp-4 (Efficiency Comparison of GCT *and* Hybrid*)*. In this experiment, we compare GCT with a very competitive method Hybrid. As a hybrid approach of partial answer saving and online search, Hybrid keeps in advanced the top-r vertices for all possible k and r. For an input query of parameters k and r, Hybrid can directly get the answer of top-r vertices and then computes the corresponding social contexts using Algorithm 1 in an online manner. The main cost of Hybrid is the social context computation. Fig. 9 shows the running time of Hybrid and GCT on three datasets by varying r from 1 to 300 and $k = 3$. Hybrid is comparative to GCT when $r = 1$. However, when r goes larger, GCT is significantly faster than Hybrid on all datasets, which reflects the superiority of our GCT-index-based diversity search.

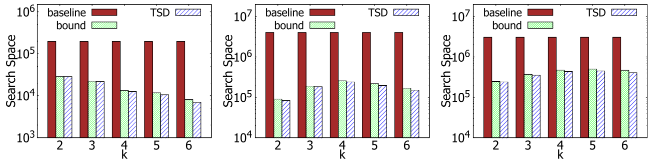
7.2 Effectiveness Evaluation

This experiment evaluates the effectiveness of truss-based structural diversity model for social contagion. As mentioned in the introduction, social contagion is an information diffusion process that a user of a social network gets affected by the information propagated from his/her neighbors. In this experiment, we simulate the social contagion by the process of influence propagation using the independent cascade model [4], [15]. In the independent cascade model, vertices in the input graph have two state:



(a) Gowalla (b) LiveJournal (c) Orkut

Fig. 7. Comparsion of baseline, bound, Core-Div, Comp-Div and TSD in terms of running time (in seconds).



(a) Gowalla (b) LiveJournal (c) Orkut

Fig. 8. Comparsion of baseline, bound, and TSD in terms of search space.

TABLE 3
Comparison of TSD and GCT Indexing Methods in Terms of the Index Size, Index Construction Time, and Query Time

Network	Graph Size	Index Size		Index Construction Time		Query Time	
		TSD	GCT	TSD	GCT	TSD	GCT
Wiki-Vote	1.1MB	4.2MB	4MB	9.82s	8.45s	7.0ms	1.8ms
Email-Enron	3.9MB	7.2MB	5.6MB	10.80s	8.82s	18.2ms	5.5ms
Epinions	5.4MB	13.3MB	13.1MB	35.36s	25.79s	31.9ms	6.3ms
Gowalla	21MB	34.9MB	29.7MB	49.24s	30.17s	70.2ms	23.7ms
NotreDame	20MB	45.4MB	19.8MB	286s	223s	106ms	65.4ms
LiveJournal	478MB	1,670MB	1,352MB	9,297s	6,689s	4.9s	1.2s
socfb-konect	1,510MB	663MB	106MB	1,603s	629s	6s	1.6s
orkut	1,130MB	4,090MB	3,812MB	16,012s	9,819s	10.7s	1.7s

unactivated and activated. Initially, we apply influence maximization algorithm [34] on graph G to obtain 50 vertices as a set of activated seeds. Then we uses these seeds to influence their neighbors. If one of their neighbors get activated from the previous unactivated status, we say that this vertex gets contagion. For a activated seed u and its unactivated neighbor v, the successful activation of v from u only depends on the edge probability between u an v. We perform the Monte Carlos sampling for 10,000 times. Then, we evaluate the number of target vertices (output by different approaches) that get activated (social contagion) by these seeds in the influence propagation. We treat undirected graphs as directed graphs, by regarding each undirected edge $e = (u, v)$ as two directed edges $< u, v >$ and $< v, u >$, with the same influential probability $p(e) = 0.01$ by default.

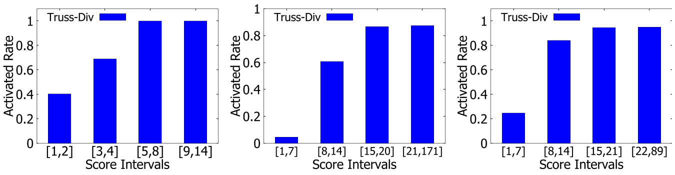
Exp-5 (Correlation Between Social Contagion and Truss-Based Structural Diversity). This experiment attempts to validate the correlation between social contagion and truss-based structural diversity. We test whether the vertices with higher truss-based structural diversity scores would have higher probabilities to get activated. We set the parameter $k = 4$. According to the scores of truss-based structural diversity, we partition the vertices into 4 groups with different score intervals from low to high. We report the activated rate of each group, that is, the number of activated vertices over the total number of vertices in this group. Fig. 10 reports the activated rates of all groups on three networks of Gowalla, LiveJournal, and Orkut. The results show that the vertices having higher scores are more easily to get activated. It confirms that truss-based structural diversity is a good predictor for social contagion.

Exp-6 (Effectiveness Comparison of Different Models). We apply all competitor methods Random, Comp-Div, Core-Div, and our method Truss-Div to obtain r vertices, by setting the parameter $k = 4$ if necessary. We evaluate how many vertices among those top-r vertices selected by different methods will get activated in the influence propagation. The larger the number of activated vertices is, the better is. Fig. 11 shows the number of activated vertices by different methods varied by parameter r. We can see that our method has more number of activated vertices than all the other methods, indicating the vertices with larger truss-based structural diversities have a higher probability to get affected by others.

Exp-7 (Latency Incurred to Activate the Results of Different Models). This experiment evaluates the latency (the number of activation rounds) incurred to activate the top-100 results of Truss-Div, Core-Div and Comp-Div. Fig. 12 reports the

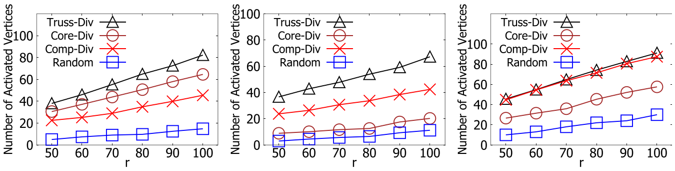
TABLE 4
Running Time (in Seconds) of TSD and GCT for Ego-Network Extraction and Ego-Network Truss Decomposition

Network	Ego-network Extraction Time		Ego-Network Truss Decomposition Time	
	TSD	GCT	TSD	GCT
Wiki-Vote	3.5s	**2.2s**	6.6s	**4.5s**
Email-Enron	4.4s	**2.2s**	5.8s	**3.9s**
Epinions	14s	**6.7s**	18.8s	**11s**
Gowalla	31.2s	**8.53s**	16.1s	**11.8s**
NotreDame	49.2s	**18.5s**	226s	**160s**
Livejournal	1,094s	**663s**	7,902s	**5,240s**
socfb-konect	1,399s	**135s**	78.2s	**75.4s**
orkut	7,180s	**2,469s**	7,350s	**4,349s**



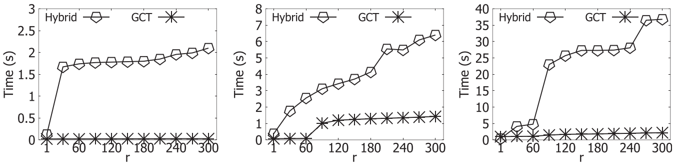
(a) Gowalla (b) LiveJournal (c) Orkut

Fig. 10. Correlation between social contagion and truss-based structural diversity.



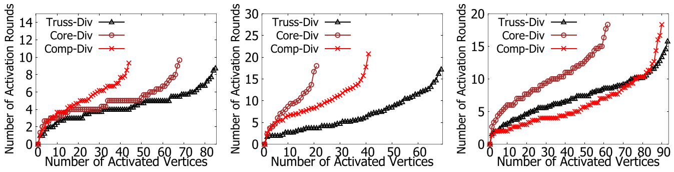
(a) Gowalla (b) LiveJournal (c) Orkut

Fig. 11. Comparison of Random, Comp-Div, Core-Div, and TSD in terms of the number of activated vertices.



(a) Gowalla (b) LiveJournal (c) Orkut

Fig. 9. Running time (in seconds) of Hybrid and GCT varied by r.



(a) Gowalla (b) LiveJournal (c) Orkut

Fig. 12. Latency of activating top-100 results by three models.

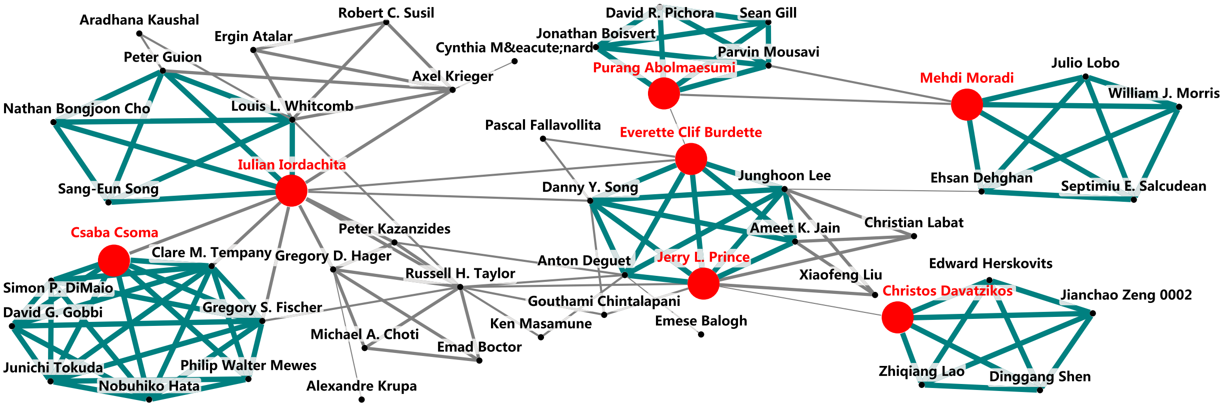


Fig. 13. A case study of structural diversity search on DBLP. Here, $k = 5$ and $r = 1$. This is an ego-network of "Gabor Fichtinger". Each component in green is a maximal connected 5-truss, which represents a distinct social context.

average number of activation rounds w.r.t the number of activated vertices on three networks. Truss-Div achieves the smallest latency to activate the most number of vertices on Gowalla and Livejournal. Truss-Div is competitive with Comp-Div on Orkut, due to the imbalanced structural diversity distribution of top-100 results of Comp-Div. The activated speed of Comp-Div gets fast first and then slows down significantly. It shows that the vertices selected by Truss-Div are more quickly and easily to get social contagion than the Core-Div and Comp-Div models.

7.3 Case Study on DBLP

We conduct a case study on a collaboration network from DBLP.[2] The DBLP network consists of 234,879 vertices and 542,814 edges. An author is represented by a vertex. An edge between two authors indicates that they have co-authored for at least 3 times. We make a comprehensive comparison of Truss-Div, Comp-Div and Core-Div models on the case studies of DBLP network.

Exp-8 (Top-1 Result by Our Truss-Based Model). We set $r = 1$ and $k = 5$ to obtain the author v^* whose name is "Gabor Fichtinger" with the highest structural diversity $\mathsf{score}(v^*) = 6$. The ego-network $G_{N(v^*)}$ of "Gabor Fichtinger" visualized in Fig. 13 consists of six maximal connected 5-trusses in green, which represent six semantic contents (e.g., 6 research groups working on different topics). In contrast, we apply Comp-Div and Core-Div on this same ego-network $G_{N(v^*)}$ and obtain the following *meaningless results*. For Comp-Div, the whole network cannot be decomposed into multiple social contexts using the component-based model for any k-sized component [6]. For Core-Div, in Fig. 13, the six components are connected together to form a connected 4-core through the edges between the authors highlighted in red.

Hence, it is also difficult to apply the Comp-Div and Core-Div models for effective structural diversity analysis on this complex ego-network $G_{N(v^*)}$. This further shows the superiority of truss-based structural diversity model on the analysis of large-scale complex ego-networks.

Exp-9 (Quality Evaluation of Social Contexts). Table 5 reports the statistics of three ego-networks of top-1 result by Comp-Div, Core-Div, and Truss-Div on DBLP. We report the author name of answers, vertex size, edge size, density, the number of social contexts (i.e., $|\mathsf{SC}(v)|$), and activated probability. We perform influence propagation in the network formed by each top-1 result v and its neighbors. We assign the edge probability to 0.05 uniformly, and randomly select 10 influential seeds from $N(v)$. The top-1 result of Truss-Div achieves the highest activated probability of 0.47 on the average of 10,000 runs, which verifies the superiority of our truss-based structural diversity model. Moreover, the ego-network of "Gabor Fichtinger " by Truss-Div has the largest density of 5.18.

8 Related Work

Structural Diversity Search. Social decisions can significantly depend on the social network structure [12], [14]. Ugander *et al.* [35] conducted extensive studies on the Facebook to show that the contagion probability of an individual is strongly related to its structural diversity in the ego-network. Motivated by [35], Huang *et al.* [19] studies the problem of structural diversity search to find k vertices with the highest structural diversity in graphs. To improve the efficiency of [19], Chang *et al.* [6] proposes a scalable algorithm by enumerating each triangle at most once in constant time. Structural diversity search based on a different k-core model is further studied in [18]. The k-truss-based structural diversity studied in this work is also called k-brace-based structural diversity [35]. In addition, there also exist numerous studies on *top-k query processing* [1], [3], [30], [37] by considering diversity in the returned ranking results. However, the problem of structural diversity search based on k-truss model has not been investigated by any study mentioned above.

K-Truss Mining and Indexing. In the literature, there exist a large number of studies on k-truss mining and indexing. As a cohesive subgraph, k-truss requires that each edge has at least $(k-2)$ triangles within this subgraph [9]. Interestingly, several equivalent concepts of k-truss termed as different names are independently studied. For example, k-truss has

TABLE 5
Ego-Network Statistics of Top-1 Results on DBLP

Methods	Author Name (ego)	$\|V\|$	$\|E\|$	Density	$\|\mathsf{SC}(v)\|$	Activated Probability
Comp-Div	Ming Li	130	344	2.64	8	0.44
Core-Div	Rui Li	38	148	3.89	3	0.43
Truss-Div	Gabor Fichtinger	51	264	**5.18**	6	**0.47**

[2] https://dblp.uni-trier.de/xml

been named as the k-dense community [16], [31], k-mutual-friend subgraph [39], k-brace [35], and triangle k-core [38]. The task of truss decomposition is to find the non-empty k-truss for all possible k's in a graph. Wang and Cheng [36] propose a fast in-memory algorithm for truss decomposition. In addition, truss decomposition has also been studied in various computing settings (e.g., external-memory algorithms [36], MapReduce algorithms [7], and shared-memory parallel systems [32]) and different types of graphs (e.g., uncertain graphs [13], [22], directed graphs [33], and dynamic graphs [20], [38]).

Recently, several community models are built on the k-truss [2], [20], [21], [40]. Meanwhile, a number of k-truss-based indexes (e.g., TCP-index [20] and Equi-Truss [2]) are proposed for another problem of community search, which supports the efficient retrieval of communities. In contrast to the above studies, k-truss-based structural diversity search is first studied in this paper. Leveraging the micro-network analysis of ego-networks, we propose a novel tree-shaped structure of TSD-index and efficient algorithms to address our problem. A detailed of index comparison can be found in [17].

9 Conclusion

In this paper, we investigate the problem of truss-based structural diversity search over graphs. We propose a truss-based structural diversity model to discover social contexts, which has a strong decomposition to break up weak-tied social groups in large-scale complex networks. We propose several efficient algorithms to solve the top-r truss based structural diversity search problem. We first develop efficient techniques of graph sparsification and an upper bound for pruning. We also propose a well-designed and elegant TSD-index for keeping the information of structural diversity which solves the problem in time linear to graph size. Moreover, we develop a new GCT algorithm based on GCT-index. Experiments also show the effectiveness and efficiency of our proposed truss-based structural diversity model and algorithms, against state-of-the-art component-based and core-based methods.

Acknowledgment

This work was partially supported by NSFC 61702435, RGC 12200917, 12201018, CRF C6030-18GF, and Guangdong Science and Technology Special Fund SDZX2019042.

References

[1] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong, "Diversifying search results," in *Proc. 2nd ACM Int. Conf. Web Search Data Mining*, 2009, pp. 5–14.

[2] E. Akbas and P. Zhao, "Truss-based community search: A truss-equivalence based indexing approach," *Proc. VLDB Endowment*, vol. 10, no. 11, pp. 1298–1309, 2017.

[3] A. Angel and N. Koudas, "Efficient diversity-aware search," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2011, pp. 781–792.

[4] S. Bian, Q. Guo, S. Wang, and J. X. Yu, "Efficient algorithms for budgeted influence maximization on massive social networks," *Proc. VLDB Endowment*, vol. 13, no. 9, pp. 1498–1510, 2020.

[5] R. S. Burt, "Social contagion and innovation: Cohesion versus structural equivalence," *Amer. J. Sociol.*, vol. 92, no. 6, pp. 1287–1335, 1987.

[6] L. Chang, C. Zhang, X. Lin, and L. Qin, "Scalable top-k structural diversity search," in *Proc. IEEE 33rd Int. Conf. Data Eng.*, 2017, pp. 95–98.

[7] P.-L. Chen, C.-K. Chou, and M.-S. Chen, "Distributed algorithms for k-truss decomposition," in *Proc. IEEE Int. Conf. Big Data*, 2014, pp. 471–480.

[8] N. Chiba and T. Nishizeki, "Arboricity and subgraph listing algorithms," *SIAM J. Comput.*, vol. 14, no. 1, pp. 210–223, 1985.

[9] J. Cohen, "Trusses: Cohesive subgraphs for social network analysis," *Nat. Secur. Agency Tech. Rep.*, vol. 16, pp. 3–29, 2008.

[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. Cambridge, MA, USA: MIT Press, 2009.

[11] F. Ding and Y. Zhuang, "Ego-network probabilistic graphical model for discovering on-line communities," *Appl. Intell.*, vol. 48, no. 9, pp. 3038–3052, 2018.

[12] Y. Dong, R. A. Johnson, J. Xu, and N. V. Chawla, "Structural diversity and homophily: A study across more than one hundred big networks," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2017, pp. 807–816.

[13] F. Esfahani, J. Wu, V. Srinivasan, A. Thomo, and K. Wu, "Fast truss decomposition in large-scale probabilistic graphs," in *Proc. Int. Conf. Extending Database Technol.*, 2019, pp. 722–725.

[14] J. H. Fowler and N. A. Christakis, "Cooperative behavior cascades in human social networks," *Proc. Nat. Acad. Sci. USA*, vol. 107, pp. 5334–5338, 2010.

[15] A. Goyal, W. Lu, and L. V. Lakshmanan, "CELF++: Optimizing the greedy algorithm for influence maximization in social networks," in *Proc. Int. Conf. World Wide Web*, 2011, pp. 47–48.

[16] E. Gregori, L. Lenzini, and C. Orsini, "k-Dense communities in the internet AS-level topology," in *Proc. Int. Conf. Commun. Syst. Netw.*, 2011, pp. 1–10.

[17] J. Huang, X. Huang, and J. Xu, "Truss-based structural diversity search in large graphs," 2020, *arXiv: 2007.05437*.

[18] X. Huang, H. Cheng, R. Li, L. Qin, and J. X. Yu, "Top-k structural diversity search in large networks," *VLDB J.*, vol. 24, no. 3, pp. 319–343, 2015.

[19] X. Huang, H. Cheng, R.-H. Li, L. Qin, and J. X. Yu, "Top-k structural diversity search in large networks," *Proc. VLDB Endowment*, vol. 6, no. 13, pp. 1618–1629, 2013.

[20] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu, "Querying k-truss community in large and dynamic graphs," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2014, pp. 1311–1322.

[21] X. Huang and L. V. Lakshmanan, "Attribute-driven community search," *Proc. VLDB Endowment*, vol. 10, no. 9, pp. 949–960, 2017.

[22] X. Huang, W. Lu, and L. V. Lakshmanan, "Truss decomposition of probabilistic graphs: Semantics and algorithms," in *Proc. Int. Conf. Manage. Data*, 2016, pp. 77–90.

[23] R. R. Huckfeldt and J. Sprague, *Citizens, Politics and Social Communication: Information and Influence in an Election Campaign*. Cambridge, U.K.: Cambridge Univ. Press, 1995.

[24] D. Kempe, J. M. Kleinberg, and É. Tardos, "Maximizing the spread of influence through a social network," in *Proc. 9th ACM SIGKDD Int. Conf. Knowl. Discov. Data Mining*, 2003, pp. 137–146.

[25] M. Latapy, "Main-memory triangle computations for very large (sparse (power-law)) graphs," *Theor. Comput. Sci.*, vol. 407, no. 1–3, pp. 458–473, 2008.

[26] J. Leskovec and A. Krevl, "SNAP datasets: Stanford large network dataset collection," Jun. 2014. [Online]. Available: http://snap.stanford.edu/data

[27] J. Mcauley and J. Leskovec, "Discovering social circles in ego networks," *ACM Trans. Knowl. Discov. Data*, vol. 8, no. 1, 2014, Art. no. 4.

[28] R. Pastor-Satorras and A. Vespignani, "Epidemic spreading in scale-free networks," *Phys. Rev. Lett.*, vol. 86, no. 14, 2001, Art. no. 3200.

[29] J. Qin, Y. Chen, W. Fu, Y. Kang, and M. M. Perc, "Neighborhood diversity promotes cooperation in social Dilemmas," *IEEE Access*, vol. 6, pp. 5003–5009, 2018.

[30] L. Qin, J. X. Yu, and L. Chang, "Diversifying top-k results," *Proc. VLDB Endowment*, vol. 5, no. 11, pp. 1124–1135, 2012.

[31] K. Saito, T. Yamada, and K. Kazama, "Extracting communities from complex networks by the k-dense method," *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, vol. 91, no. 11, pp. 3304–3311, 2008.

[32] S. Smith, X. Liu, N. K. Ahmed, A. S. Tom, F. Petrini, and G. Karypis, "Truss decomposition on shared-memory parallel systems," in *Proc. IEEE High Perform. Extreme Comput. Conf.*, 2017, pp. 1–6.

[33] T. Takaguchi and Y. Yoshida, "Cycle and flow trusses in directed networks," *Roy. Soc. Open Sci.*, vol. 3, no. 11, 2016, Art. no. 160270.

[34] Y. Tang, Y. Shi, and X. Xiao, "Influence maximization in near-linear time: A martingale approach," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2015, pp. 1539–1554.

[35] J. Ugander, L. Backstrom, C. Marlow, and J. Kleinberg, "Structural diversity in social contagion," *Proc. Nat. Acad. Sci. USA*, vol. 109, no. 16, pp. 5962–5966, 2012.

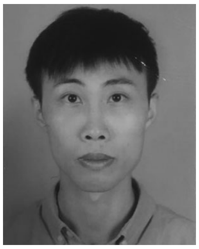
[36] J. Wang and J. Cheng, "Truss decomposition in massive networks," *Proc. VLDB Endowment*, vol. 5, no. 9, pp. 812–823, 2012.

[37] Y. Zhang, J. Callan, and T. Minka, "Novelty and redundancy detection in adaptive filtering," in *Proc. 25th Annu. Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval*, 2002, pp. 81–88.

[38] Y. Zhang and S. Parthasarathy, "Extracting analyzing and visualizing triangle k-core motifs within networks," in *Proc. IEEE 28th Int. Conf. Data Eng.*, 2012, pp. 1049–1060.

[39] F. Zhao and A. K. Tung, "Large scale cohesive subgraphs discovery for social network visual analysis," *Proc. VLDB Endowment*, vol. 6, pp. 85–96, 2012.

[40] Z. Zheng, F. Ye, R.-H. Li, G. Ling, and T. Jin, "Finding weighted k-truss communities in large networks," *Inf. Sci.*, vol. 417, pp. 344–360, 2017.



Jinbin Huang received the bachelor's degree in computer science from the South China University of Technology (SCUT), Guangzhou, China. He is currently working toward the PhD degree at Hong Kong Baptist University (HKBU), Hong Kong.

Xin Huang received the PhD degree from the Chinese University of Hong Kong (CUHK), Hong Kong, in 2014. He is currently an assistant professor with Hong Kong Baptist University. His research interests mainly focus on graph data management and mining.

Jianliang Xu received the PhD degree from the Hong Kong University of Science and Technology, Hong Kong. He is currently a professor with the Department of Computer Science, Hong Kong Baptist University. He is an associate editor of the *IEEE Transactions on Knowledge and Data Engineering* and the *Proceedings of the VLDB Endowment* 2018.