Journal of Big Data

RESEARCH Open Access

Discovering top-weighted k-truss communities in large graphs

Wafaa M. A. Habib[1*], Hoda M. O. Mokhtar[1,2] and Mohamed E. El-Sharkawi[1,2]

*Correspondence:
w.momen@fci-cu.edu.eg
[1] Information Systems Department, Faculty of Computers and Artificial Intelligence, Cairo University, Cairo, Egypt
Full list of author information is available at the end of the article

Abstract

Community Search is the problem of querying networks in order to discover dense subgraphs-communities-that satisfy given query parameters. Most community search models consider link structure and ignore link weight while answering the required queries. Given the importance of link weight in different networks, this paper considers both link structure and link weight to discover top-r weighted k-truss communities via community search. The top-weighted k-truss communities are those communities with the highest weight and the highest cohesiveness within the network. All recent studies that considered link weight discover top-weighted communities via global search and index-based search techniques. In this paper three different algorithms are proposed to scale-up the existing approaches of weighted community search via local search. The performance evaluation shows that the proposed algorithms significantly outperform the existing state-of-the-art algorithms over different datasets in terms of search time by several orders of magnitude.

Keywords: Community search, Weighted graph, k-truss community detection model

Introduction

Community search is a major problem in graph model which had recently gained excessive attention from researchers. Community Search problem is to search a graph to discover a community that satisfies certain query parameters. For example, a community that contains a certain vertex or a set of keywords is required to be discovered. There are many studies over community search especially on large graphs [1–9]. Most of studies within community search usually ignore edge weight. The edge weight is playing an important role where it is used to represent the strength of the relationship between any two vertices. There are many applications that clarify the importance of edge weight:

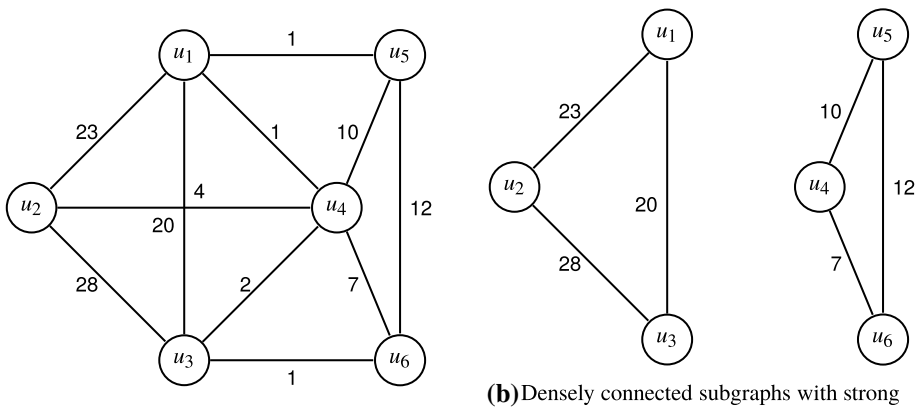
- The edge weight in a co-authorship network may indicate how many papers the two linked authors had co-authored together [10]. Considering the edge weight during community search would ensure that authors within discovered communities have strong co-authoring relationship between them.

- The edge weight in social network may represent the similarity, or interactions between users [10]. Considering the edge weight in the resulted community would ensure the discovery of highly interacted and similar group of users.
- Corporate ownership networks (CON), this is a weighted economic network that links 406 different countries, and its weights represent the business ties among countries [11]. Considering the edge weight within the discovered communities would reveal and ensure the business ties between countries.

A sample of social network is illustrated in Fig. 1 where vertices represent users and edge between any two vertices represents the friendship relation. In such social network the edge weight plays an important role describing the social interactions between users. The two communities in Fig. 1a and in Fig. 1b are densely connected in terms of the number of edges between vertices. Besides, the two communities in Fig. 1b show the top interacting groups of users based on the weights on the edges between them. The community with minimum edge weight 20 is considered as the top weighted community of interacting users while the community with minimum edge weight 7 is considered as the second top weighted community. In this example, edge weight was able to distinguish between different groups of users according to their interaction level. In addition, all three communities in Fig. 1a and b are densely connected where their trussness is equal to three. The trussness level of three ensures that every two connected nodes have one common neighbor and consequently ensures a high level of structural similarity between nodes.

Inspired by the importance of edge weight, this paper considers the edge weight to discover weighted communities. More specifically, the proposed models in this paper utilize edge weight and k-truss model in order to discover top weighted k-truss communities. k-truss community is the densest subgraph in which each edge resides in at least k-2 triangles [1], where triangle models the cyclic relationship between 3 vertices.

Querying top weighted k-truss communities has been studied recently in the literature using different methods like online method and index based method [10]. Both methods discovers top weighted k-truss communities using global search where the whole graph



(a) Densely connected graph.

(b) Densely connected subgraphs with strong authorship.

Fig. 1 Motivation example

resides in the main memory and all edges are required to be visited. Another direction to discover top weighted communities utilizes local search method [4, 12]. Local search discovers the community of a given vertex using the neighbouring vertices and their edges. Local search is more efficient than global search as it searches a small portion of the graph to discover the required communities. On the other hand, local search techniques proposed in the literature don't consider edge weigh and discover coherent k-core local communities only. The main challenge of local search while considering edge weight is to find the small portion of the graph that certainly contains the required top weighted communities.

This paper builds on the concept of local search in order to obtain the same output communities discovered by global search. The utilization of local search technique is motivated by their search strategy that tends to check the neighborhood of a node rather than checking the whole graph. Using such a search strategy to obtain the same results obtained by global search would guarantee an extremely less search time while having the required results. This paper utilizes local search in three different methods in order to optimize the required search time to find the top-weighted k-truss communities. Local search is performed by checking the vicinity of the edge with the highest weight while processing edges in a descending order based on their weights. Once a community is found in the vicinity of the edge being checked between the edge and its neighbors, a local solution can be confirmed. The local solution is an evidence for the existence of-at least-one global solution within the portion of the graph checked so far. Then, the global solutions can be found by checking this portion of the graph instead of the whole graph.

The main contribution of this paper is utilizing local search technique in three proposed algorithms to discover top-weighted k-truss communities. More specifically, the main contributions are as follow:

- A *LOCAL k-TRUSS ALGORITHM (LKA)*: is proposed as a base solution to apply local search. The algorithm processes edges with the highest weight in sequence. With each edge, the vertices of the edge and their common neighbouring that are processed so far are checked to find out if they form a local k-truss community.
- A *DEGREE-BASED LOCAL k-TRUSS ALGORITHM (DBLKA)*: based on the fact that any k-truss community must be *(k-1)-core* community and not vice versa, *DBLKA* is proposed. Similar to *LKA*, edges with the highest weight are processed in sequence. With each edge, the vertices of the edge and their common neighbouring that are processed so far are checked to find out if they form a *(k-1)-core* first before checking if they form a local k-truss community. Consequently, the number of local communities that need to be checked for k-truss existence are decreased which improves search time as shown in "Performance evaluation" section.
- A *MULTIPLE CANDIDATE LOCAL k-TRUSS ALGORITHM (MCLKA)*: Similar to *DBLKA*, the algorithm check the existence of *(k-1)- core* before checking the the existence of local k-truss. On the other hand, the algorithm does the checking process on multiple edges vertices and their common neighbours at once rather than checking each edge case on its own. The collective checking process for multiple candidates has led to a dramatic improvement in terms of search time as shown in "Performance evaluation" section

For all proposed algorithms; once the number of local k-truss communities reaches the required number of communities, the edges processed so far are examined using enumeration algorithm [10] in order to find global communities resides in them.

The rest of this paper is organized as follows: "Related work" section presents related work. "Preliminaries" section overviews some of the basic concepts used in the paper including weighted graphs, edge support, and weighted k-truss communities. In "Proposed algorithms" section the proposed algorithms are presented. "Performance evaluation" presents the empirical results and discusses them. Finally, "Conclusion" section concludes the paper and highlights possible directions for future work.

Related work

There are several models in the literature which address the problem of discovering cohesive subgraph in terms of structure which is called community detection task. Community detection models [13–16] are used to discover group of vertices that are strongly connected to each others and weakly connected to outside vertices. The most commonly used and familiar techniques to discover dense subgraphs are cliques [17, 18], quasi-clique [19], k-core [20, 21, 21], edge density [22, 23], edge connectivity [24, 25], and k-truss [1, 2]. Recently, authors in [26] proposed a new model called KTMiner to detect k-truss communities in a distributed manner using Map-Reduce framework on Apache Spark environment.

Community search is another task where the goal is to discover cohesive group of vertices but in terms of search according to a certain query in the graph rather than detection of all existing communities. Community search is proposed to address the problem of discovering group of vertices that contains a specific vertex, set of vertices, or set of keywords. The community search problem is well studied in the literature [4–6, 27–30]. A global search procedure is proposed by the authors in [27] to search for a subgraph that contains a query vertex by iteratively removing vertices with the minimum degree which can be computed in a linear time. In [4] an efficient local search procedure for the same problem is proposed by the authors where the algorithm starts from the vertex query and expand the search to its neighbours in order to find the best community that query vertex resides into. A novel α-adjacency γ-quasi-k-clique model was proposed by the authors in [28] to study the overlapping community search problem. In [5, 29], the community search problem is studied by utilizing the *k-truss* model, where the maximal connected *k-truss* component containing a query vertex is considered as a community.

Another category of community search algorithms are weight-based community search. Influential community search is an example of weight-based community search where the each node in the graph is weighted with its influence. In [3], two algorithms;online and index are proposed to ed discover the top weighted influential communities where k-core model is utilized. In [8], the authors extended online algorithm of[3] in two ways namely Backward and Forward algorithms. Backward algorithms it starts the search by adding vertices with the highest weight and verify the component if it is a k-core or not; if it is a k-core a solution is returned otherwise it proceeds to add more vertices. Forward algorithm iteratively removes vertices with the minimum weight until the graph becomes disconnected and the top communities

are returned. The authors in [12] proposed local-optimal algorithm which considered is the state of art according to its performance. Local-optimal algorithm build its search space incrementally by adding subsets of vertices with the highest weights until the required top weighted communities are discovered. All these techniques are node-weight based which utilize the *k-core* model as their cohesion measure. Another weight-based community search algorithm is proposed in [10] which utilize k-truss as its cohesion model. It differs from other algorithms as it considers an edge-weighted approach rather than node-weighted. The authors proposed two different techniques to retrieve top weighted k-truss communities, the first one is discovering communities online; The procedure starts by discovering the maximal k-truss of the original graph, and iteratively removes edges with the minimum weight and with each removal a maximal connected component procedure is run to find the next k-truss connected component. The online approach cannot scale for large graph as the whole graph should be resides in main memory. The other approach is the index based where all the weighted k-truss communities are indexed separately for each k. The required communities are returned directly from the index. The index based approach is more efficient than the online base one, it suffers from the large size of the index which requires much time to traverse that index. In addition, the maintenance of the index would be time consuming. Recently authors in [31] proposed two online algorithms namely *BACKWARD ALGORITHM*, and *WEIGHT-SENSITIVE LOCAL SEARCH ALGORITHM (WSLSA)*. The main idea for the two proposed algorithms is iteratively attaching the edge with the highest weight. The two proposed algorithms are, the *BACKWARD ALGORITHM*, and *WEIGHT-SENSITIVE LOCAL SEARCH ALGORITHM (WSLSA)* overcome the drawbacks of the online search algorithms proposed in [10]. The *BACKWARD ALGORITHM* algorithm detects the *top-r weighted k-truss* communities by iteratively attaching the edges with the highest weight after reducing the graph to its k-truss. On the other hand, the *WEIGHT-SENSITIVE LOCAL SEARCH ALGORITHM (WSLSA)* detects the *top-r* communities by visiting only the highest weighted edges in the graph without the need to reduce the graph into its k-truss. The drawback of *BACKWARD ALGORITHM* and *WEIGHT-SENSITIVE LOCAL SEARCH ALGORITHM (WSLSA)* is their failure to process large graphs especially when k approaches the max level of trussness. When k approaches the max level of trussness, more candidate solutions have to be verified where candidate size gets bigger with each cycle. Consequently, the algorithm fails when it has to verify candidates with very large size. *k-core* is a community detection model which discovers a connected subgraph where each vertex has degree no less than k. Another community detection model is called $k-truss$ which is defined based on the concept of triangle where each edge in a connected $k-truss$ subgraph resides in at least $k-2$ triangles. All previous models that addressed the problem of top weighted community search have focused mainly on global search solutions on a way or another. Except for *BACKWARD ALGORITHM* and *WEIGHT-SENSITIVE LOCAL SEARCH ALGORITHM (WSLSA)*, all models have either used a global search or built an index structure to reduce search time. On the other hand, models presented in [31] utilize local search but suffer from limitations while processing large graphs. The local

search paradigm utilized in this paper presents an opportunity to perform community search while having the least search time.

Preliminaries

Table 1 describes the notions that are utilized within the paper. Given undirected and edge weighted graph $G(V, E, W)$, where V, E, and W represent vertices, edges, and vector of weights respectively. Each entry in the vector of weights is assigned a different weight value for each edge. Each edge is denoted by $e(u, v)$, its weight denoted by $\omega(e)$, and the set of neighbors of a vertex υ are denoted by $nb(\upsilon)$, *i.e.*, $nb(\upsilon) = \{ u \in \mathrm{V}: \exists \mathrm{e}\ (u, \upsilon) \in \mathrm{E}\}$, and degree of υ is denoted by $\mathrm{d}(\upsilon) = |nb(\upsilon)|$. A triangle denoted by $\bigtriangleup_{uvw}$ is a cyclic relationship between three vertices u, v, w such that $(u, v), (u, w), (v, w) \in E$. Given an induced subgraph $H(V_H, E_H)$ from G where $V_H \subseteq V_G$ and $E_H \subseteq E_G$, the support of an edge $e(u, v) \in H$ is defined as the number of triangles that edge resides in, and denoted by $sup(e, H)$. The edge trussness is the edge support increased by 2.

Definition 1 *(k-truss)* A subgraph $H(V_H, E_H, W_H)$ is a connected k-truss iff each e_H has $sup(e, H)$ at least k-2. A subgraph H is called a maximal k-truss if there is no other subgraph $H\prime$ contains H.

The trussness of a subgraph H denoted as $\tau(H)$ is the minimum support of the all edges in subgraph H incremented by 2, e.g. $\tau(\mathrm{H}) = \min\{\mathrm{sup}(e, \mathrm{H}): e \in \mathrm{E_H}\} + 2$.

In this paper weighted graph is considered where the weight of the subgraph H is defined as the minimum weight of the set of edges in subgraph H.

Definition 2 Subgraph Weight: The weight of subgraph H denoted by $f(H)$ is the minimum weight of the edges-weights in H, e.g. $f(H) = min_{\mathrm{e} \in E_H}\{\omega(e)\}$. The edge with minimum weight in subgraph H is called the key-edge of H.

The rational behind the minimum weight is that each edge in the subgraph H has at least this minimum weight as discussed in [3]. In addition, minimum weight would be robust to outliers than average weight.

Table 1 Frequently used notations

Notation	Description
$G = (V, E, W)$	Undirected and edge weighted graph
$\mathrm{n} = \lvert V \rvert, \mathrm{m} = \lvert E \rvert$	Number of vertices and number of edges
$\mathrm{n}_X = \lvert V_X \rvert, \mathrm{m}_X = \lvert \mathrm{E}_X \rvert$	Number of vertices and number of edges in subgraph X
$nb(v)$	The set of neighbors of v
$\mathrm{d}(v)$	The degree of v
$sup(e,H)$	The support of edge e in subgraph H
$\omega(e)$	The weight of edge e
$f(\mathrm{H})$	The weight of subgraph H $\min_{e \in \mathrm{E_H}}\{\omega(e)\}$
$\mathrm{S}_{(u,v)}$	The common neighbors of vertex (u, v), $nb(u) \cap nb(v)$
$\tau(H)$	The trussness of subgraph H
$\mathrm{E}_{S(u,v)} \leftrightarrow u, v$	The set of edges between $_{S(u,v)}$ and $\{u, v\}$

Based on the previous definitions of k-truss and subgraph weight, the weighted k-truss community is defined as follow.

Definition 3 Weighted k-truss Community: Given undirected and edge weighted graph $G = (V, E, W)$, and trussness level k, a subgraph H $\subset G$ is weighted k-truss community satisfies the following constraints:

- *Connectivity* H is a connected subgraph
- *k-truss* The minimum sup(e,H) is at least k-2.
- *Maximal* There is no other subgraph H' contains H and the $f(H')=f(H)$.

By applying the three conditions of weighted k-truss community while extracting resulting communities, the output communities are guaranteed to be k-truss and not a subset from other weighted k-truss community with the same weight.

Example 1 Consider the graph in Fig. 2. Suppose for instance $k = 4$, as clarified in Definition 3 the original graph is a weighted 3-truss community with minimum weight of value = 1. In addition, two weighted communities with higher weights reside in the original graph; the top-1 4-truss subgraph shown in Fig. 2 highlighted by a red rectangle with weight value 37 of the edge $e(v_1, v_3)$. The highlighted subgraph by green rectangle shown in the same Fig. 2 is the top-2 4-truss community with weight 35 of the edge $e(v_2, v_4)$. The subgraph induced by the set of edges $\{(v_1, v_2), (v_2, v_3), (v_1, v_3), (v_3, v_4), (v_1, v_4), (v_2, v_4)\}$ also has weight 35; however it is not weighted $4 - truss$ community sine it is already contained in the subgraph highlighted by green rectangle with the same weight 35.

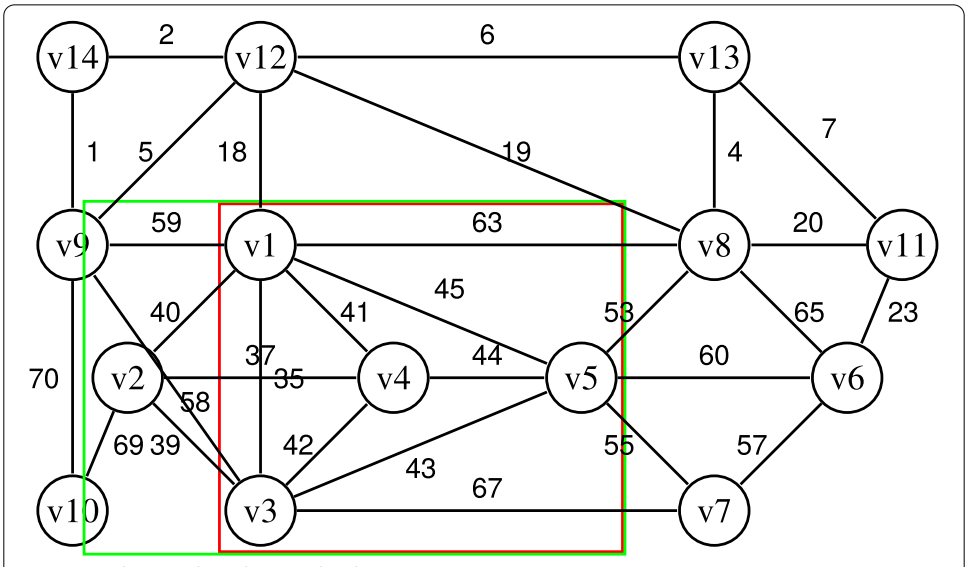


Fig. 2 Graph example and its weighted communities

Problem Definition $G = (V, E, W)$ is an undirected and edge-weighted graph where r and k are the two query parameters. The problem is defined as the task to discover the *top-r weighted k-truss* communities from $G = (V, E, W)$

Example 2 Consider the example illustrated in Fig. 2, suppose k = 4 and r = 2, the top-2 weighted 4-truss communities are highlighted in red and green rectangles in Fig. 2. The top-1 4-truss community is the one highlighted by red rectangle with weight 37 where each edge in the community resides in two triangles. The top-2 4-truss community shown in the same Figure highlighted by green rectangle with weight value 35. The top-2 4-truss community contains the top-1 community but it has smaller weight than the top-1 community.

Proposed algorithms

This section discusses the proposed algorithms to discover *top-r weighted k-truss* communities. k-truss community detection model is used to measure the cohesiveness of the resulting communities. Since k-truss is defined based on the concept of triangle; *k-truss* model main advantage is related to its ability to ensures a high level of cohesiveness. In addition, a community with certain *k-truss* is also a community with *(k-1)-core* on the same time but not vice versa which guarantees the high cohesiveness level of *k-truss*. *k-truss* community is a *(k-1)-core* community since it is *(k-1)*-edge connected and any deletion of no fewer than k-1 edges will not disconnect *k-truss*. Also, *k-truss* is a diameter bounded algorithm where a subgraph of n vertices has a diameter no more than $[2n - 2/k]$. All these properties are indicators for the cohesivness of the resulting communities from the the k-truss model [32]. For self completeness of this paper truss decomposition algorithm introduced in [2] is outlined in Algorithm 1.

Algorithm 1 Truss Decomposition

Input: Graph G(V, E), k
Output: k-class of the Graph
1:
2: compute sup(e,G) for each $e \in E_G$
3: **for each** $(e(u, v)$ such that $sup(e, G) < k - 2)$ **do**
4: assume w.l.o.g. $deg(u) < deg(v)$;
5: **for each** $w \in nb(u) and e(v, w) \in E_G$ **do**
6: $sup((u, w)) \leftarrow sup((u, w)) - 1$;
7: $sup((v, w)) \leftarrow sup((v, w)) - 1$;
8: $Reorder\ (u, w), and\ (v, w)\ according\ to\ their\ new\ support$
9: **end for**
10: remove $e(u, v)$ from G;
11: **end for**
12: **if** (not all edges $e \in E_G$ not removed) **then**
13: assign all remaining edges into H with $\tau(H) = k$
14: **end if**

A local search procedure is used in the proposed algorithms. Mainly a local vicinity of the edge e with highest weight $w(e)$ is built from the edges with weights higher then $w(e)$ and checked whether it is a connected k-truss component or not. Suppose that two vertices u, v, and an edge $e(u, v)$ is considered as the edge being checked, a local vicinity of this edge is built from the set of edges between common neighbors of u and v and $e(u, v)$ itself. A local vicinity is defined as follow:

Definition 4 local vicinity $H_{e(u,v)}$ is the set of edges $E_{S(u,v)} \leftrightarrow u, v$ where each edge E_i in E is defined as $(v_1, v_2) : v_1 \in S_{u,v}, v_2 \in u, v$ and $S_{(u,v)} = nb(u) \cap nb(v)$ is the common neighbors of u, v.

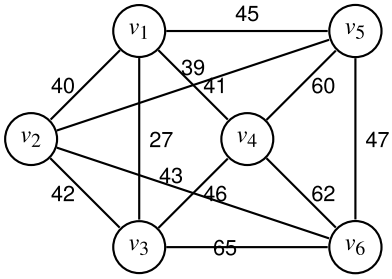
Local Vicinity H of the edge is checked whether it has connected k-truss component iff sup(e,H) at least k-2, $|nb(u) \cap nb(v)|$=k and degree of each vertex in $S_{(u,v)}$ at least k-1 which is formalized in 1.

Property 1 *A k-truss connected component should have at least k vertices with degree at least k-1.*

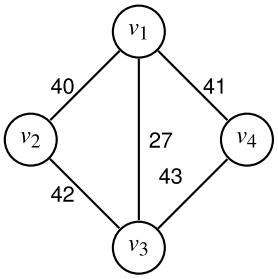
This paper argues that the existence of a $k - truss$ component in the local subgraph of edge vicinity would prove the existence of at least one global community that would be discovered by decomposing the larger subgraph containing the local graph. The existence of community in the local subgraph is used as an evidence that the edges processed so far has a community within them. Thus, In the first step; the edges with the highest weights are processed until the required number of communities is discovered through local communities subgraphs. Afterwards, global communities can be discovered by decomposing the subgraph containing only the edges that was processed during the first step. Such procedure would allow the proposed algorithms to discover the top weighted communities while processing a small subset of the edges with the highest weights rather than the whole graph edges. Formally, we define the paper argument in the following property followed by an example to illustrate the idea of local communities versus global communities.

Property 2 *Given a local vicinity subgraph $H \subseteq G$, if there is a k-truss $H' \in H$ then there is k-truss component $H'' \in G$ and $H' \subseteq H''$.*

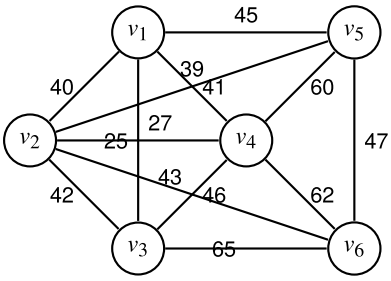
Example 3 Consider the graph G shown in Fig. 3a contains set of the highest weight edges where $e(v_1, v_3)$ is the one with the least weight of 27 and considered for local vicinity subgraph checking. Property 1 is satisfied within the local vicinity of edge $e(v_1, v_3)$ and the local subgraph is built as shown in Fig. 3b where the goal is to extract top weighted 4-truss communities. However, the local subgraph has no 4-truss community. Then, the next edge $e(v_2, v_4)$ with weight 25 is considered for local vicinity subgraph checking . Property 1 is satisfied within the local vicinity of of edge $e(v_2, v_4)$ and the local subgraph is built as shown in Fig. 3d. The local subgraph is found to have a local 4-truss community. Thus, the local exploration of the edges has led to the discovery of one local community. However, the graph in its current form as shown in Fig. 3c has two top weighted global communities that can be discovered only through k-truss decomposition of graph 3c. The two top weighted global communities are themselves the two graphs in Fig. 3a and Fig. 3c with weights 27 and 25. The local graph in Fig. 3b didn't have a local community where the graph in Fig. 3a that the local subgraph was extracted from has a global community. This example shows that the local exploration of the edges local vicinity can provide an evidence for communities existence rather than truly discovering them. In addition, the existence of one local community is an evidence of the existence of at least one global community where multiple global communities can exist in the original



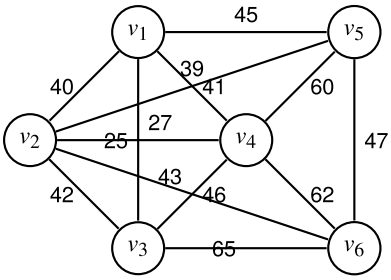
(a) Graph G until edge $e(v_1, v_3)$ with weight 27.



(b) Local Community Verification for edge $e(v_1, v_3)$.



(c) Graph G until edge $e(v_2, v_4)$ with weight 25.



(d) Local Community Verification for edge $e(v_2, v_4)$.

Fig. 3 Local community versus Global community Example

graph. Thus, it is a necessity to analyze the graph that local graphs are extracted from in order to discover the set of global communities.

This paper proposes three algorithms which utilize property 1 and property 2 to discover *top-r weighted k-truss* communities; *LOCAL k-TRUSS ALGORITHM (LKA), DEGREE-BASED LOCAL k-TRUSS ALGORITHM (DBLKA), MULTIPLE CANDIDATE LOCAL k-TRUSS ALGORITHM (MCLKA)* where each of them is explained in the following subsections.

Local k-truss algorithm

This algorithm consists of three main steps, all three main steps work together to eventually define a subgraph y that contains the required top-r weighted communities. Through this way, only a small subgraph y can be used to discover the communities

rather than the original graph G. The three main steps are listed below and explained in more details:

- Build temp graph.
- Temp graph verification.
- Enumerating the required *top-r weighted k-truss* communities.

Local k-truss algorithm takes as an input the weighted graph G along with k, r; The weighted graph G edges are presorted in descending order and stored on disk. The algorithm starts by processing the edges with the highest weight in sequence. The processed set of edges are added into subgraph Y. With each edge being processed, the effect of this edge is checked according to property 1. If the edge $e(u, v)$ has k-2 common neighbors i.e. $|S_{u,v} \geq k-2|$ and each with degree at least $k-1$ in subgraph Y, a k-truss component is suspected. Then, a temp graph X is constructed, $X = E_{S_{(u,v)}} \leftrightarrow \{u, v\}$ where the set of vertices are the two vertices u, v in addition to the set of vertices of $S_{(u,v)} = nb(u) \cap nb(v)$ that reside in subgraph Y. Finally, edges between $S_{(u,v)}$ in Y are added to temp graph X.

The next step—Temp graph verification—is to check if the temp graph X has a k-truss component. The algorithm impose count triangle procedure in order to remove edges with support less than k-2. Then, detect connected component procedure is run to check whether there is a k-truss connected component in X or not as outlined in Algorithm 1. Once a community in X is discovered, the number of verified communities is incremented by 1. The decomposition of X is expected to be fast in terms of time as the decomposition is done only on a small number of edges in subgraph X.

Finally, once the number of verified local communities reach the required number of the k-truss weighted communities r, the global communities should be discovered from subgraph y as realization of property 2. The enumeration procedure firstly decomposes the subgraph y that contains r-communities and iteratively removes edges with the minimum weight that represent the key-edges of the community. Before each key-edge removal, a community with the weight of the key-edge is retrieved and stored. The removal of the edges continues till graph y is no longer connected. Then, top r weighted communities are retrieved and considered as the output. The pesudo code of *LOCAL k-TRUSS ALGORITHM (LKA)* is outlined in Algorithm 2.

Algorithm 2 `Local k-truss Algorithm (LKA)`

Input: $Graph\ G(V, E, W), k, and\ r$
Output: r communities
1: assume $Graph\ G(V, E, W)$ is presorted w.r.t. edge weights in descending order;
2: **while** ($\exists e_{(u,v)} \in E_G$ and $num\_local\_Communities < r$) **do**
3: add $e_{(u,v)}$ into Y;
4: **if** ($e_{(u,v)}$ is not pruned due to property 0.1) **then**
5: $X_{e(u,v)} \leftarrow Build\_Temp\_Graph(X_{e(u,v)})$;
6: **end if**
7: $H \leftarrow TrussDecomposition(X_{(u,v)}, k)$; ▷ make k-truss of X as in Algorithm 1
8: **if** (H is not empty) **then**
9: $num\_local\_communities+ = 1$;
10: **end if**
11: **end while**
12: $Enumerating\_Global\_Communities(Y)$

Complexity analysis Local k-truss Algorithm complexity can be described as the decomposition cost of each candidate subgraph in addition to the decomposition cost of the subgraph containing the global solutions. Given the number of candidates subgraphs as α, the edges in each candidate subgraph X as m_x and the decomposition cost of edges as the number of edges to the power of 1.5 as mentioned in [33], the complexity of the first phase of the algorithm (from line 2 to line 10 in Algorithm 2) can be formally defined as $O(\alpha * m_x^{1.5})$. Similarly, the decomposition of the subgraph Y containing the global solutions (line 12 in algorithm 2) would be $O({m_y}^{1.5})$. The total complexity of the Local k-truss Algorithm would be $O(\alpha * m_x^{1.5}) + O(m_y^{1.5})$.

Example 4 This example shows a detailed explanation for local k-truss algorithm where $k = 4$ and $r = 2$ are considered as the search parameters. The subgraph Y contains the highest weight edges processed so far from graph G in Fig. 2, upon processing edge $e(v_1, v_3)$, property 1 is satisfied where $e(v_1, v_3)$ has four vertices $\{v_2, v_4, v_5, v_9\}$ as a common neighbours with degree 3. Then, temp graph X for $e(v_1, v_3)$ is built as shown in Fig. 4b. Afterwards, decompose k-truss procedure is run over $X_{(v1,v3)}$ where the subgraph highlighted by green circle will be removed as the support of all its edges is equal to 1. The subgraph highlighted by red rectangle is a *4-truss* community. Figure 4c shows the same subgraph Y where the edge $e(v_2, v_4)$ is added and checked according to property 1 the temp graph $X_{(v2,v4)}$ is built as shown in Fig. 4d and a $4-truss$ is found. Thus,

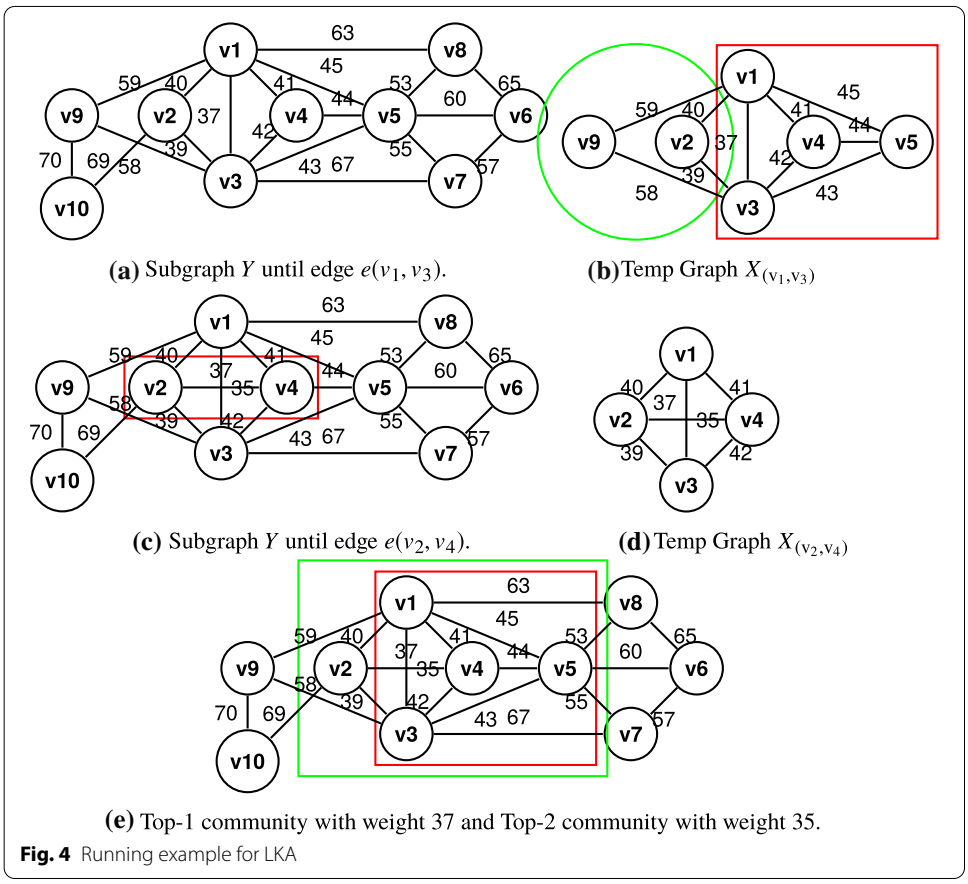


Fig. 4 Running example for LKA

subgraph Y with edge $e(v_2, v_4)$ has at least two global 4-truss communities according to property 2 as two local communities are discovered. The top-2 4-truss communities are highlighted in Fig. 4e, the top-1 4-truss community is highlighted by7 red rectangle and top-2 4-truss community is highlighted by green rectangle.

Degree based local k-Truss Algorithm *(DBLKA)*

In this subsection *DBLKA* is proposed to extract local communities more efficiently. It follows the same steps similar to *LKA* except for the verification step which introduces *k-core* filtration as an extra step during the verification process. The three main steps are listed below and explained in more details:

- Build temp graph.
- Temp graph verification.
- Enumerating the required *top-r weighted k-truss* communities.

All these three steps work together to mainly extract *top-r weighted* communities. The *top-r weighted k-truss* communities are extracted from a subgragh Y rather than original graph G. The *DBLKA* takes as an input original graph G, k, and r where the set of edges are presorted in descending order in terms of edge weights. The set of edges are processed iteratively starting from the edge with the highest weight. It starts with building the temp graph X following the same steps as explained in the first step in section "Local k-truss algorithm". After the temp graph X is built a verification step is performed. The main idea of the verification step here is based on the fact that any $k-truss$ component must be $(k-1)-core$ component. Therefore, the temp graph X is checked if it is $(k-1)-core$ before performing k-truss decomposition. K-truss decomposition is performed only if temp graph X is found to be $(k-1)-core$. Otherwise, the edge being processed is ignored and next edge with the highest weight is considered for processing.

Temp graph X is checked if it is $(k-1)-core$ by removing any vertex with degree less than $k-1$. Thus, the remaining vertices if any represents a $(k-1)-core$ component. If a $(k-1)-core$ component is found, its vertices and edges are checked against two different cases; First, if the number of the remaining vertices are greater than k then the remaining vertices and their edges should be processed to check if they form a $k-truss$ component or not. Second, if the number of remaining vertices are equal to k vertices with degrees equal to $k-1$ then the remaining vertices with their edges is a $k-clique$ component and consequently a $k-truss$ component.

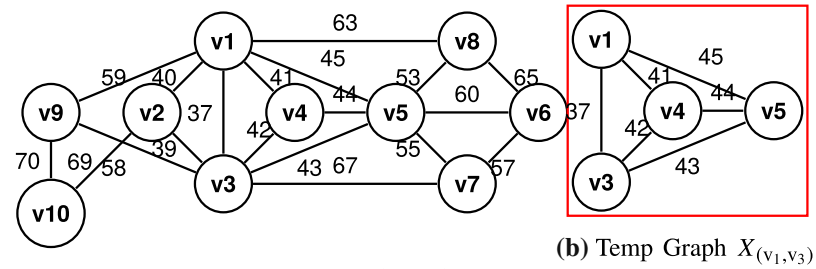
Algorithm 3 *Degree Based Local k-truss Algorithm (DBLKA)*

Input: $Graph\ G(V, E, W)$, k, and r.
Output: *top-r weighted k-truss communities.*
1: assume $GraphG(V, E, W)$ is presorted w.r.t. edge weights in descending order;
2: **while** ($\exists e_{(u,v)} \in E_G$ and $num\_local\_Communities$<r) **do**
3: add $e_{(u,v)}$ into Y;
4: **if** ($e_{(u,v)}$is not pruned due to property 0.1) **then**
5: $X_{\text{e(u,v)}} \leftarrow Build\_Temp\_Graph(X_{e(u,v)})$;
6: **end if**
7: **if** ($X_{\text{e(u,v)}}$) is $(k-1)-core$) **then**
8: $num\_vertices \leftarrow count\_vertices \in X$;
9: **if** ($num\_vertices$==k) **then**
10: $num\_local\_communities+ = 1$
11: **end if**
12: **if** ($num\_vertices > k$) **then**
13: $H \leftarrow TRussDecomposition(X_{(u,v)}, k)$; ▹ make k-truss of X as in Algorithm 1
14: **if** (H is not empty) **then**
15: $num\_local\_communities+ = 1$;
16: **end if**
17: **end if**
18: **end if**
19: **end while**
20: $Enumerating\_Global\_Communities(Y)$

Finally the third step is performed to extract the $top-r\ weighted\ k-truss$ communities from the subgraph Y following the same steps as explained in the third step in section "Local k-truss algorithm". The pesudo code of *Degree Based LOCAL k-TRUSS ALGORITHM (DBLKA)* is outlined in Algorithm 3 and a detailed explanation for the algorithm is showed in Example 5.

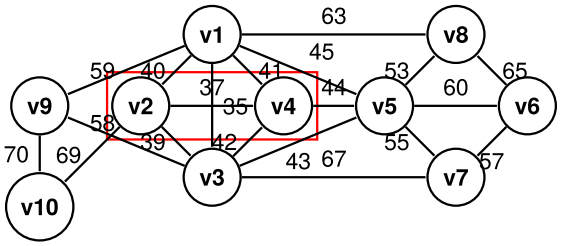
Complexity analysis DBLKA complexity can be described as the decomposition cost of each candidate subgraph in addition to the decomposition cost of the subgraph containing the global solutions. Given the number of degree-based filtered candidates subgraphs as β, the edges in each candidate subgraph X as m_X and the decomposition cost of edges as the number of edges to the power of 1.5 as mentioned in [33], the complexity of the first phase of the algorithm (from line 2 to line 19 in Algorithm 3) can be formally defined as $O(\beta * m_x^{1.5})$. Similarly, the decomposition of the subgraph Y containing the global solutions (line 20 in algorithm 3) would be $O(m_y^{1.5})$. The total complexity of the Degree Based Local k-truss Algorithm would be $O(\beta * m_x^{1.5}) + O(m_y^{1.5})$. It's noted that the number of degree-based filtered candidates subgraphs β is expected to be less than the number of candidates subgraphs α generated by the Local k-truss Algorithm.

Example 5 Consider the graph G in Fig. 2 where $k = 4$ and $r = 2$ are considered as the search parameters. The subgraph Y contains the highest weight edges processed so far, upon processing edge $e(v_1, v_3)$, property 1 is satisfied where $e(v_1, v3)$ has four vertices $\{v_2, v_4, v_5, v_9\}$ as a common neighbours with degree 3. Then, temp graph X for $e(v_1, v_3)$ is built as shown in Fig. 5b where the two vertices $\{v_2, v_9\}$ are excluded while building X due to $k-1$ core filtration step. The the two vertices $\{v_2, v_9\}$ are not a part from $3-core$ where the degree of each them is 2. The remaining component is highlighted by red in Fig. 5b is a $4-truss$ community without calling decompose k-truss procedure as $X_{(v1,v3)}$ contains 4 vertices with degree 3 which turns it to as a clique clarified in the proposed

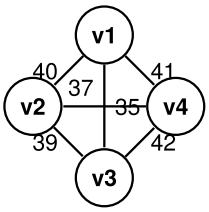


(a) Subgraph Y until edge $e(v_1, v_3)$.

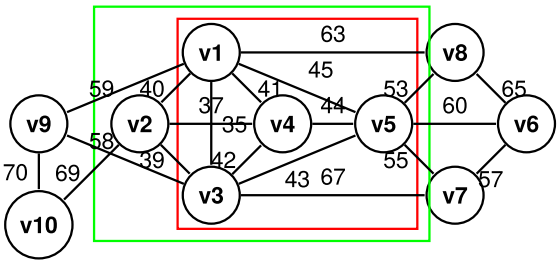
(b) Temp Graph $X_{(v_1,v_3)}$ after degree filtration.



(c) Subgraph Y until edge $e(v_2, v_4)$.



(d) Temp Graph $X_{(v_2,v_4)}$.



(e) Top-1 community with weight 37 and Top-2 community with weight 35.

Fig. 5 Running example for DBLKA

algorithm. Figure 5c shows the same subgraph Y where the edge $e(v_2, v_4)$ is added and checked according to property 1. Then, the temp graph $X_{(v2,v4)}$ is built as shown in Fig. 5d and a $4-$*truss* is found directly given the number of vertices and their degrees where no vertices will be removed in $k-1$ core filtration step and the remaining number of vertices with their degrees form a clique as clarified in the proposed algorithm. Thus, subgraph Y with edge $e(v_2, v_4)$ has at least two global 4-truss communities according to property 2 as two local communities are discovered. The top-2 4-truss communities are highlighted in Fig. 5e, the top-1 4-truss community is highlighted by red rectangle and top-2 4-truss community is highlighted by green rectangle.

Multiple candidates local k-truss algorithm*(MCLKA)*

In this subsection the third algorithm *MCLKA* is proposed where multiple temp graphs of multiple edges with the highest weight and satisfying property 1 are generated as one multiple candidates graph. The generated multiple candidates graph is verified once at a time instead of verifying each temp graph separately. In generating multiple candidates graph, edges with the highest weight are processed in sequence where the vertices and their common neighbours of each edge satisfying property 1 is added to multiple

candidates graph Z. Once the number of processed edges satisfying property 1 reach r, temp graph Z is built and its vertices are connected. Then, a verification step is performed over the multiple candidates graph Z. The verification of multiple candidates at once should allow a faster processing for the edges and consequently a faster discovery for the *top* − *r weighted k* − *truss* communities. The three main steps are listed below and explained in more details:

- Generate Multiple Candidates Graph.
- Multiple Candidates Graph verification.
- Enumerating the required *top-r weighted k-truss* communities.

Algorithm 4 *Multiple Candidates Local k-truss Algorithm (MCLKA)*

Input: $Graph\ G(V, E, W)$, k, and r.
Output: *top-r weighted k-truss communities.*
Input: $Graph\ G(V, E, W)$, k, and r.
Output: *top-r weighted k-truss communities.*
assume $GraphG(V, E, W)$ is presorted w.r.t. edge weights in descending order;
while ($\exists e_{(u,v)} \in E_G$ and $num\_local\_Communities < r$) **do**
  add $e_{(u,v)}$ into Y;
  if ($e_{(u,v)}$ is not pruned due to property 0.1) **then**
    $List\_key - edges \leftarrow e_{(u,v)}$;
    add w into affected_vertices : $\forall w \in \{nb(u) \cap nb(v)\}$ and $deg(w) \geq k - 1$;
  end if
  if ($len(List\_key - edges) < r$) **then**
    go to step3;
  end if
  if ($len(List\_key - edges) == r$) **then**
    $Z \leftarrow Build\_Temp\_Graph(affected\_vertices)$
  end if
  $H \leftarrow TrussDecomposition(Z, k)$; ▷ make k-truss of X as in Algorithm 1
  if (H is not empty) **then**
    while (**do**$\exists e \in List\_key - edges$ and $e \in H$)
      $num\_local\_Communities$ += 1;
    end while
    empty $List\_key - edges$
  end if
end while
$Enumerating\_Global\_Communities(Y)$

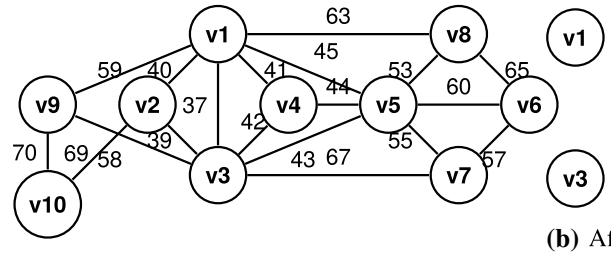
In the first step—Generate Multiple Candidates Graph—the edges are processed in sequence from the highest weight edge where the processed set of edges are added into subgraph Y. For each edge $e(u, v)$ being processed, the edge is checked to find out if it satisfies property 1 or not. If the edge was found to be satisfying for property 1, the set of affected vertices with degree >= k−1; the two vertices u, v in addition to the set of vertices of $S_{(u,v)}$=$nb(u) \cap nb(v)$ that reside in subgraph Y are added to graph Z. In addition, $e(u, v)$ is considered as a key-edge for the set of the affected vertices that was added to subgraph Z and saved in key-edges list for later processing. The processing of the edges continue until the number of key edges reaches r. Once the number of key-edges reaches r, the edges between vertices in graph Z which exist in graph Y are added to graph Z.

In the second and the third step—Multiple Candidates Graph verification and Enumerating the required *top-r weighted k-truss* communities, the subgraph Z is verified to check whether there is a connected $k - truss$ component in it or not. The count

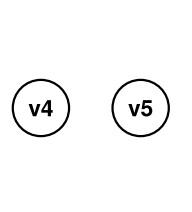
triangle procedure is performed over all multiple candidates in subgraph Z to remove any edge with support less than $k-2$. If a $k-truss$ component is found, then the existence of the key-edges from the key-edges list are checked in the decomposed version of subgraph Z. If r key-edges are found in decomposed version of subgraph Z, then the subgraph Y is decomposed to extract and enumerate *top-r weighted k-truss* communities. On the other hand, if less than r key-edges existed on the $k-truss$ component or no $k-truss$ component was found in the decomposed version of subgraph Z, the algorithm gets back and starts to process edges with the highest weights for another cycle. The pesudo code of *Multiple Candidates Local k-truss Algorithm (MCLKA)* is outlined in Algorithm 4 followed by a detailed explanation in Example 6.

Complexity analysisMCLKA complexity can be described as the decomposition cost of each multiple candidate subgraph in addition to the decomposition cost of the subgraph containing the global solutions. Given the number of multiple candidate subgraphs as μ, the edges in each multiple candidate subgraph X as m_X and the decomposition cost of edges as the number of edges to the power of 1.5 as mentioned in [33], the complexity of the first phase of the algorithm (from line 2 to line 21 in Algorithm 4) can be formally defined as $O(\mu * m_x^{1.5})$. Similarly, the decomposition of the subgraph Y containing the global solutions (line 22 in Algorithm 4) would be $O(m_y^{1.5})$. The total complexity of Multiple Candidates Local k-Truss Algorithm would be $O(\mu * m_x^{1.5}) + O(m_y^{1.5})$. It's noted that the number of multiple candidate subgraphs μ is expected to be less than the number of degree-based filtered candidates subgraphs β generated by Degree Based Local k-truss Algorithm.

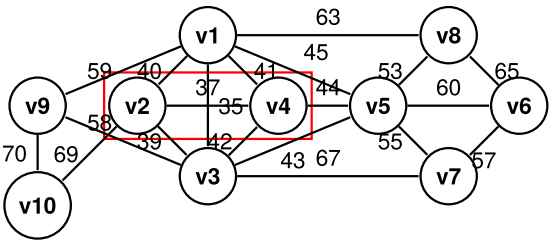
Example 6 Consider the graph G in Fig. 2 where $k=4$ and $r=2$ are the search parameters. The subgraph Y contains the highest weight edges processed so far from graph G, upon processing edge $e(v_1, v_3)$, property 1 is satisfied where $e(v_1, v_3)$ has four vertices $\{v_2, v_4, v_5, v_9\}$ as a common neighbours with degree 3. Then, the list of affected vertices of $e(v_1, v_3)$ are added to the temp graph Z as shown in Fig. 6b where the two vertices $\{v_2, v_9\}$ are excluded due to $k-1$ core filtration step. The the two vertices $\{v_2, v_9\}$ are not a part from $3-core$ where the degree of each them is 2. Figure 6c shows the same subgraph Y where the edge $e(v_2, v_4)$ is added and found to be satisfying to property 1. Accordingly, the temp graph Z is updated as shown in Fig. 6d by adding the vertex v_2 and the list of key-edges is updated by adding the current one $e(v_2, v_4)$. The temp graph Z is then updated by adding the edges between the set of vertices in graph Z as shown in Fig. 6e. The edges between vertices are extracted from graph y in Fig. 6c. The temp graph Z is decomposed where the decomposed version is found to contain the two key-edges. The existence of two key-edges in the decomposed version of Z is an evidence to decompose the subgraph Y to extract the required top-2 4-truss communities. The top-2 4-truss communities are highlighted in Fig. 6f, the top-1 4-truss community is highlighted by red rectangle and top-2 4-truss community is highlighted by green rectangle.



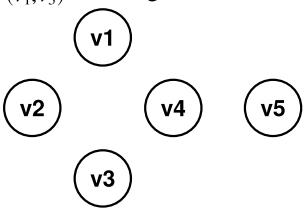
(a) Subgraph Y until edge $e(v_1, v_3)$.



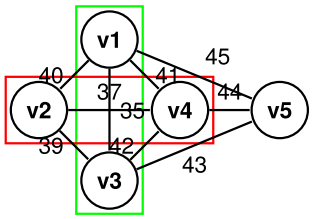
(b) Affected vertices of the key-edge $e_{(v_1,v_3)}$ after degree filtration..



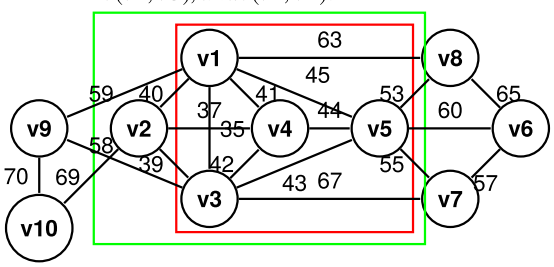
(c) Subgraph Y until edge $e(v_2, v_4)$.



(d) Updated Temp Graph Z that contains the set of affected vertices with list of key-edges $e_{(v_1,v_3)}$, and $e_{(v_2,v_4)}$.



(e) Temp Graph Z that contains $X_{(v_1,v_3)}$ and $X_{(v_2,v_4)}$ with list of key-edges $e(v1, v3)$, $and e(v2, v4)$.



(f) Top-1 community with weight 37 and Top-2 community with weight 35.

Fig. 6 Running example for MCLKA

Performance evaluation

In this section the proposed algorithms are evaluated to find out their performance in terms of execution time and prove their efficiency against the state-of-the-art algorithms. The execution time was considered as evaluation metric since it was used while evaluating similar community search models in the literature [4, 6, 8, 10, 31]. The proposed algorithms are evaluated against four different algorithms

Table 2 Datasets description

Graph	|V|	|E|	k_{max}
Wiki-Vote	8K	200K	25
Email	37K	200K	20
Youtube	1.1M	3M	19
Wiki-Talk	2.4M	5M	53
Skitter	1.7M	11M	68
LiveJournal	4M	34.6M	214
Orkut	3.1M	117.1M	78

the *BFS-based Online Search Algorithm* proposed in [10] ,the *LocalSearch-P Algorithm* proposed in [12] and both algorithms *Backward Algorithm* and *Weight Sensitive Local Search Algorithm* proposed in [31]. All algorithms discover the weighted communities in an online manner. The *BFS-based Online Search Algorithm, backward Algorithm, and Weight Sensitive Local Search algorithm* are designed to find weighted k-truss communities. The *LocalSearch-P Algorithm* utilizes the concept of local community search to discover the top-k influential communities where k-core and vertex-weight are considered. In order to ensure fairness and completeness of the experiment, *LocalSearch-P Algorithm* is implemented following the same strategy but with k-truss model and weighted edges instead of k-core model and vertex-weight. All algorithms were explained in details in related work section. All experiments are conducted on seven public datasets as shown in Table 2 and are run over Python environment. In addition, all experiments are conducted on a machine with an Intel i5 2.5GHz CPU and 8 GB main memory.

Datasets

The proposed algorithms are evaluated using seven datasets shown in Table 2 and availed in [34]. The datasets are in different sizes which range from small to large size. As shown in Table 2, each dataset has set of parameters to identify its size where V represents the number of vertices, E represents the number of the edges, and k_{max} represents the max $k-truss$ that can be extracted from the dataset. As the weighted-edge graph is considered in this paper, the weight is calculated as the common neighbors between each two vertices i.e. $w(e(u,v)) = nb(u) \cap nb(v)$. The proposed methods are not affected whether the weight is calculated or given as the weighted communities will be extracted correctly.

Experimental results

The proposed algorithms are evaluated against the BFS-algorithm, the optimal version of the LocalSearch-P algorithm and the state of the art algorithms Backward algorithm and Weight Sensetive Local Search algorithm. There are two varying query parameters where the algorithms are evaluated against them. The query parameters are K and r where k represents the trussness level and varying from 5 to largest k for each dataset and r represents the number of required top communities and varying

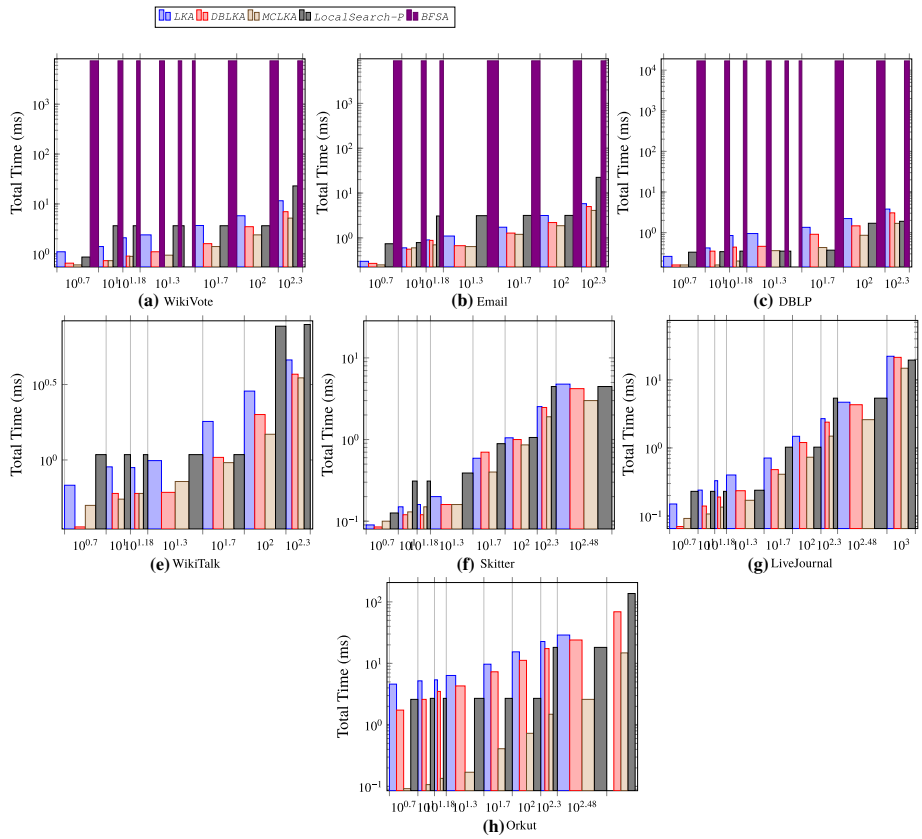


Fig. 7 Execution time of the proposed algorithms against BFS and LocalSearch-P Algorithms while k = 10 and r is varying

from 10 to 1000 in large datasets. Figures 7 and 9 show the evaluation results where k equals a default value 10 and r is varying while Figs. 8 and 10 show the evaluation results where r equals the default value 10 and k is varying. Generally Figs. 7 and 8 show that *(MCLKA)* performs better than the other algorithms. The processing time is linearly proportional to the size of the graph the algorithm visits.

With small values of both r and k, all the proposed algorithms perform the same and require the same time to discover the output communities where the algorithms usually verify a small subset of the graph. When r increases in Figs. 7 and 9, *(MCLKA)* shows a better performance as it has a faster verification process than the other two proposed algorithms *(LKA)* and *(DBLKA)*. Following the same behavior in Figs. 8 and 10 when k increases, *(MCLKA)* shows a better performance as it verifies a set of local candidates altogether at once rather than individual candidates verification. Verifying as set of local candidate saves the times required to do the verification steps for each candidate individually. Consequently, it has the ability to discover local communities with higher k efficiently than other algorithms.

As shown in Figs. 7 and 8, the BFS- algorithm takes constant time for different r as it discovers all the possible communities in the whole graph and then outputs the top ones. The BFS algorithm couldn't run for the large graphs Wiki-Talk, LiveJournal,

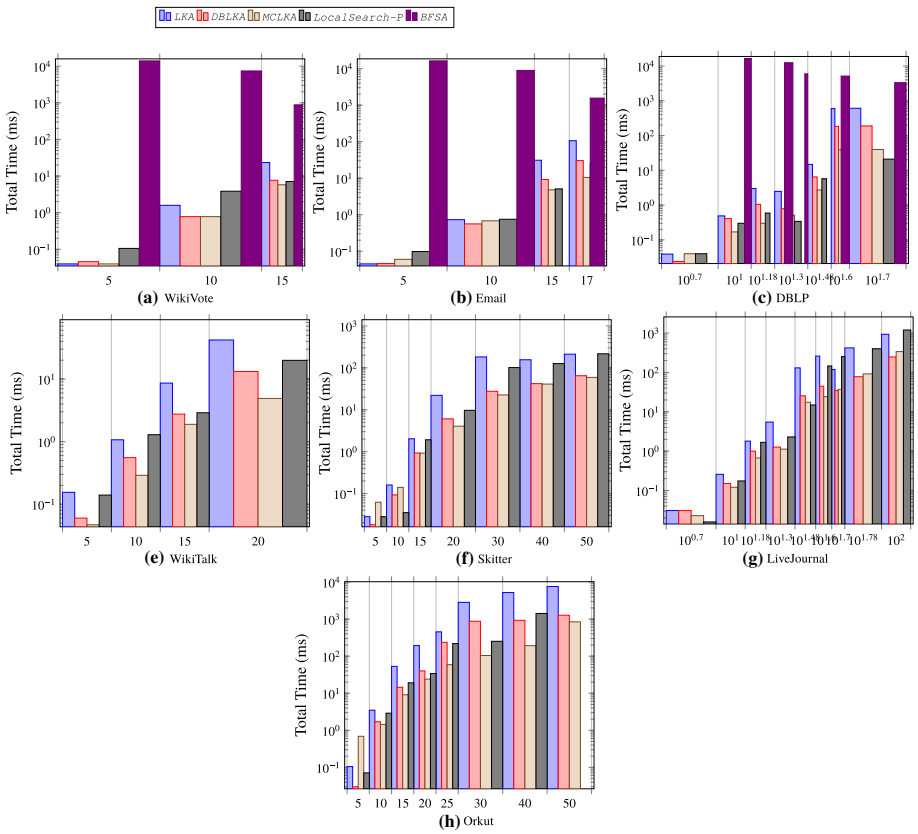


Fig. 8 Execution time of the proposed algorithms against BFS and LocalSearch-P Algorithms while r = 10 and k is varying

Skitter, and Orkut dataset due to the large size of the graphs and the inability to decompose such graphs in main memory. These graphs don't fit in a small main memory which is used during performance evaluation.

On the other hand, the localsearch-p algorithm performs in a different way where the search time increases in leaps as the search time is constant while discovering a number of communities before leaping in time to discover the next set of communities. For example, the localsearch-p algorithm examines a subset of graph that contains 100 communities while trying to discover only the top 10 weighted communities. Then, when the algorithm try to discover the top 20 weighted communities, it will also examine the same subset of the graph containing 100 communities. Consequently, the search time will be constant until the number of required communities is greater than 100. By then, the search time will have a leap as it will examine a bigger subset of the graph. The linear paradigm is better than leaps especially in large k and r.

Figures 9 and 10 indicate that *MCLKA* performs better than Backward and Weight Sensitive Local Search algorithms. *MCLKA* outperformance is related to its discovery method as it processes a small portion of graph that has the required r-communities. During this experiment, *BACWARD* has the worst performance as it first decompose the original graph into k-truss to prune all set of edges that don't reside in k-truss. *WSLSA* yeilds a close performance to *MCLKA* at different values of trussness level

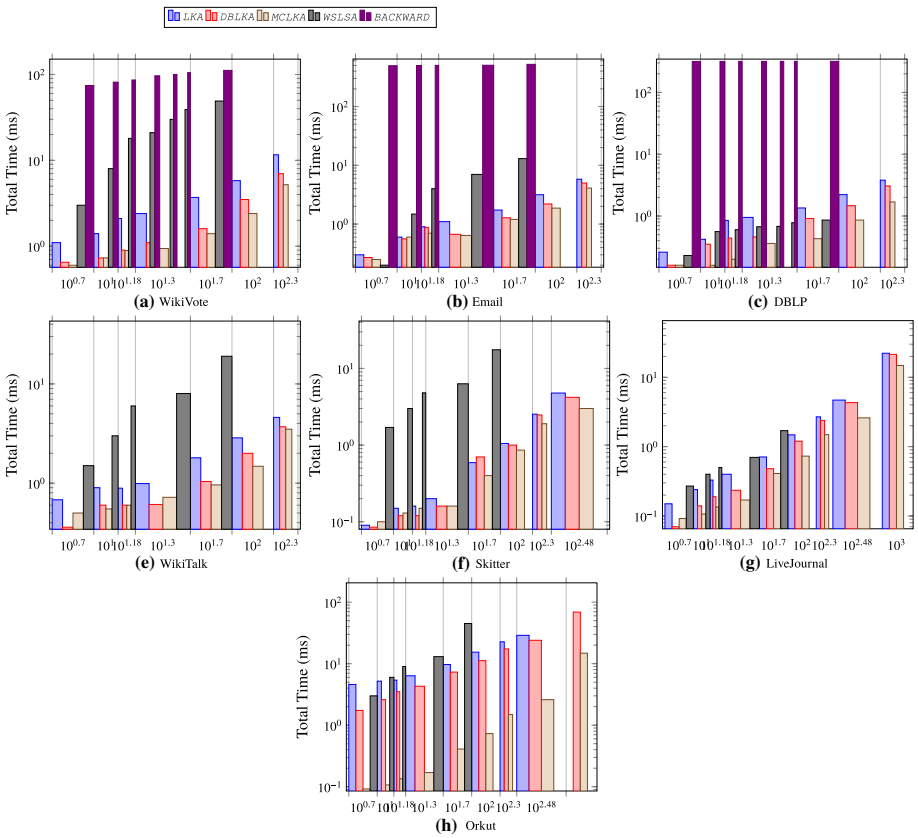


Fig. 9 Evaluation time of the proposed algorithms against Backward and WSLSA Algorithms while k = 10 and r is varying

with constant number of communities to retrieve of 10 communities. However, *WSLSA* fails at high levels of trussness unlike *MCLKA* and other local search based algorithms. Mostly, *BACWARD* and *WSLSA* algorithms failed to discover weighted communities at large values of K and large values of r unlike local-search based algorithms which did succeed at the same case.

Conclusion

In this paper community search problem is investigated to discover top-r weighted communities where the link weight is considered and the k-truss model is considered as the cohesiveness model. Three different algorithms which utilize the concept of local community search in order to discover the global community search results were proposed. The three different algorithms are *LKA*, *DBLKA*, and *MCLKA*. *LKA* where these algorithms consist of three main steps. *LKA* imposes a traditional

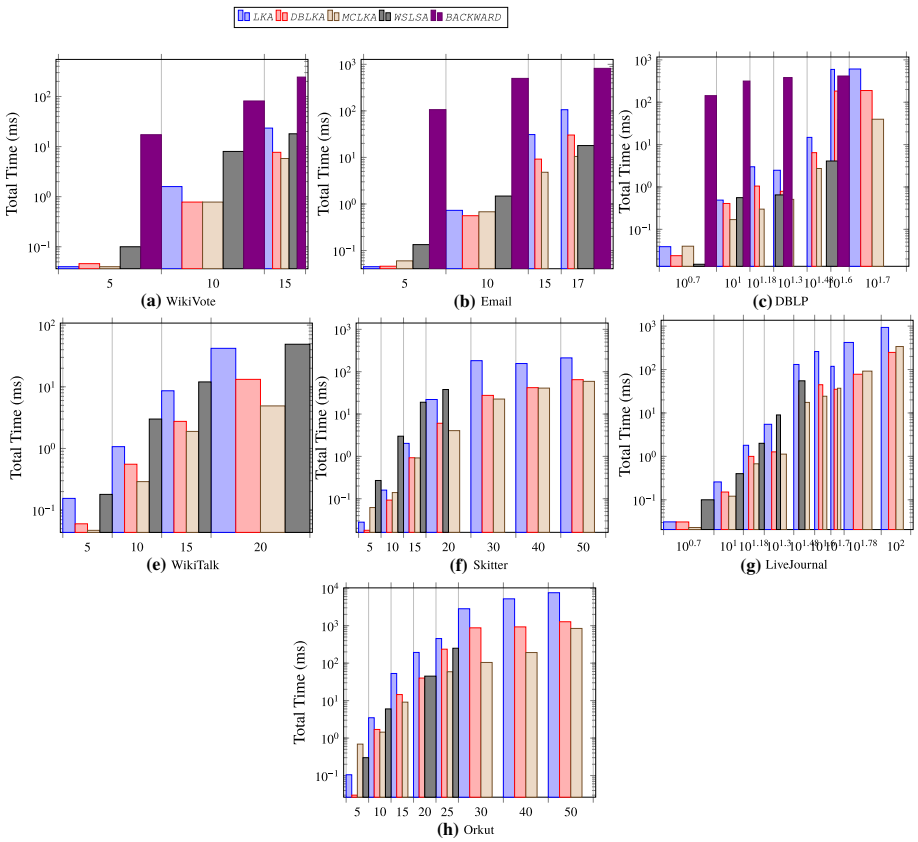


Fig. 10 Execution time of the proposed algorithms against Backward and WSLSA Algorithms while r = 10 and k is varying

procedure for k-truss model that count triangles and decompose local graphs in order to find k-truss local graphs before finding the global community search results. *DBLKA* imposes a more filtration step over the local graph to remove vertices which wont ever belong to k-truss community and consequently find k-truss local graphs in a faster way. *MCLKA* does a verification step over r-generated local graphs at once rather than considering each of them on its own. All these algorithms are evaluated against the *(BFA)*, LocalSearch-p, Backward and Weight Sensetive Local Search algorithms . Experimental results showed that *(MCLKA)* is the superior algorithm in terms of execution time against all other algorithms. One of the main challenges to consider as future work is to ensure that the difference between the number of discovered candidates and the number of truly existing communities will be low in order to enhance the search time. In addition, the proposed algorithms could be extended to add the vertices properties as an extra dimension to find homogeneous top-weighted communities. Extending the algorithms to find top weighted communities where each edge is assigned multiple weights is another future work direction.

Acknowledgements
Not applicable.

Authors' contributions
Authors contributed equal share in this research. All authors read and approved the final manuscript.

Funding
Not applicable.

Availability of data and materials
The datasets used in the experiments are available online and referenced in the paper.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Author details
[1]Information Systems Department, Faculty of Computers and Artificial Intelligence, Cairo University, Cairo, Egypt. [2]Faculty of Computing and Information Sciences, Egypt University of Informatics, Cairo, Egypt.

Received: 19 November 2021 Accepted: 23 March 2022
Published online: 03 April 2022

References

1. Cohen J. Trusses: Cohesive subgraphs for social network analysis. Natl Secur Agency Tech Rep. 2008;16:3–1.
2. Wang J, Cheng J. Truss decomposition in massive networks. Proc VLDB Endow. 2012;5(9):812–23.
3. Li RH, Qin L, Yu JX, Mao R. Influential community search in large networks. Proc VLDB Endow. 2015;8(5):509–20.
4. Cui W, Xiao Y, Wang H, Wang W. Local Search of Communities in Large Graphs. In: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data. SIGMOD '14. New York, NY, USA: ACM; 2014. p. 991–1002. http://doi.acm.org/10.1145/2588555.2612179.
5. Huang X, Cheng H, Qin L, Tian W, Yu JX. Querying k-truss community in large and dynamic graphs. In: Proceedings of the 2014 ACM SIGMOD international conference on Management of data. ACM; 2014. p. 1311–1322.
6. Akbas E, Zhao P. Truss-based community search: a truss-equivalence based indexing approach. Proc VLDB Endow. 2017;10(11):1298–309.
7. Huang X, Lakshmanan LV. Attribute-driven community search. Proc VLDB Endow. 2017;10(9):949–60.
8. Chen S, Wei R, Popova D, Thomo A. Efficient computation of importance based communities in web-scale networks using a single machine. In: Proceedings of the 25th ACM International on Conference on Information and Knowledge Management. ACM; 2016. p. 1553–1562.
9. Fortunato S. Community detection in graphs. Phys Rep. 2010;486(3–5):75–174.
10. Zheng Z, Ye F, Li RH, Ling G, Jin T. Finding weighted k-truss communities in large networks. Inf Sci. 2017;417:344–60.
11. Garas A, Argyrakis P, Rozenblat C, Tomassini M, Havlin S. Worldwide spreading of economic crisis. New J Phys. 2010;12(11): 113043.
12. Bi F, Chang L, Lin X, Zhang W. An optimal and progressive approach to online search of top-k influential communities. Proc VLDB Endow. 2018;11(9):1056–68.
13. Chang L, Li W, Qin L, Zhang W, Yang S. Fast and exact structural graph clustering. IEEE Trans Knowl Data Eng. 2017;29(2):387–401.
14. Shao J, Han Z, Yang Q, Zhou T. Community Detection Based on Distance Dynamics. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. KDD '15. New York, NY, USA: ACM; 2015. p. 1075–1084. http://doi.acm.org/10.1145/2783258.2783301.
15. Yang J, Leskovec J. Defining and evaluating network communities based on ground-truth. Knowl Inf Syst. 2015;42(1):181–213.
16. Huang X, Lu W, Lakshmanan LV. Truss decomposition of probabilistic graphs: Semantics and algorithms. In: Proceedings of the 2016 International Conference on Management of Data; 2016. p. 77–90.
17. Cheng J, Ke Y, Fu AWC, Yu JX, Zhu L. Finding maximal cliques in massive networks. ACM Trans Database Syst. 2011;36(4):21.
18. Cheng J, Zhu L, Ke Y, Chu S. Fast algorithms for maximal clique enumeration with limited memory. In: Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM; 2012. p. 1240–1248.
19. Tsourakakis C, Bonchi F, Gionis A, Gullo F, Tsiarli M. Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees. In: Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM; 2013. p. 104–112.
20. Cheng J, Ke Y, Chu S, Özsu MT. Efficient core decomposition in massive networks. In: 2011 IEEE 27th International Conference on Data Engineering. IEEE; 2011. p. 51–62.
21. Khaouid W, Barsky M, Srinivasan V, Thomo A. K-core decomposition of large networks on a single PC. Proc VLDB Endow. 2015;9(1):13–23.
22. Charikar M. Greedy approximation algorithms for finding dense components in a graph. In: International Workshop on Approximation Algorithms for Combinatorial Optimization. Springer; 2000. p. 84–95.

23. Goldberg AV. Finding a maximum density subgraph. In: Tech. Report No. UCB CSD 84/171. Computer Science Division (EECS), University of California, Berkeley, CA, 1984.
24. Chang L, Yu JX, Qin L, Lin X, Liu C, Liang W. Efficiently computing k-edge connected components via graph decomposition. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. ACM; 2013. p. 205–216.
25. Zhou R, Liu C, Yu JX, Liang W, Chen B, Li J. Finding maximal k-edge-connected subgraphs from a large graph. In: Proceedings of the 15th International Conference on Extending Database Technology. ACM; 2012. p. 480–491.
26. Alemi M, Haghighi H. KTMiner: distributed k-truss detection in big graphs. Inf Syst. 2019;83:195–216.
27. Sozio M, Gionis A. The community-search problem and how to plan a successful cocktail party. In: Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM; 2010. p. 939–948.
28. Cui W, Xiao Y, Wang H, Lu Y, Wang W. Online search of overlapping communities. In: Proceedings of the 2013 ACM SIGMOD international conference on Management of data. ACM; 2013. p. 277–288.
29. Wu Y, Jin R, Li J, Zhang X. Robust local community detection: on free rider effect and its elimination. Proc VLDB Endow. 2015;8(7):798–809.
30. Zhu Y, He J, Ye J, Qin L, Huang X, Yu JX. When Structure Meets Keywords: Cohesive Attributed Community Search. In: Proceedings of the 29th ACM International Conference on Information & Knowledge Management; 2020. p. 1913–1922.
31. Habib WM, Mokhtar HM, El-Sharkawi ME. Weight-Based K-Truss Community Search via Edge Attachment. IEEE Access. 2020;8:148841–148852.
32. Shao Y, Chen L, Cui B. Efficient cohesive subgraphs detection in parallel. In: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data; 2014. p. 613–624.
33. Latapy M. Main-memory triangle computations for very large (sparse (power-law)) graphs. Theor Comput Sci. 2008;407(1–3):458–73.
34. https://snap.stanford.edu/data/.

Publisher's Note