K-Truss Based Community Search in Weighted Graphs

Xiang Xu
The Chinese University of Hong Kong
1155107785@link.cuhk.edu.hk

Yu Chen
The Chinese University of Hong Kong
1155107766@link.cuhk.edu.hk

ABSTRACT

In real-life graph datasets, nodes may naturally have intrinsic features and the relationships between two nodes can be quantified (e.g., collaboration networks, biological networks). While a lot of existing works focus on community search on unattributed graphs, community search based on users' queries on attributed graphs is also an important research topic. In this paper, we study *community search* problem on edge-weighted graphs. In order to find closely connected communities both in structural sense and semantic sense, we formulate the problem as follows: given an edge-weighted graph, a query node q, a positive integer k and a weight threshold w, the goal is to find a k-truss community with all the triangles' weights are smaller than threshold w. Our proposed solution takes triangle as the smallest structure unit rather than using edge for measurement. We develop two different algorithms based on such strategy, one in top-down manner and the other in bottom-up manner. Experiments are conducted and presented to evaluate effectiveness and efficiency of our algorithms.

1 INTRODUCTION

Real-life scenarios involving interactions between different entities naturally have graph representations. For example, protein interaction networks and social networks are represented as graphs. Vertices within these graphs form *community structures*, which stands for groups of nodes who are closely connected with each other in a large network. Existing literature covered two important problems:

(1) *Community detection* [2, 12–14, 19] problem intends to identify community structures with interpretable meanings in a network.
(2) *Community search* problem outputs community structure consisting of the user's querying vertices [16]. A simple definition of the problem is that given query vertices in an edge-weighted graph, the goal is to find meaningful communities that the query vertices belongs to [6, 10, 16].

Existing literature introduced different data structures to tackle these problems, including quasi-clique [1], k-core [3, 4, 15] and k-truss [5, 10, 18]. Last semester, we focus on solving basic *community search* problem with k-core and k-truss by investigating into previous works.

Basic community search models place emphasis on the binary nature of the relationship between nodes, namely whether there exists an edge between two nodes. In real-life applications, graph data are more complicated. For example, the nodes may have intrinsic features and the relationships between two nodes can sometimes be quantified. This information is helpful in capturing and measuring the intimacy between nodes. Therefore, existing works also extended basic *community search* problem to different attributed graphs [9, 17, 20, 21]. In this report, we focus on *community search* problem on edge-weighted graphs due to the following reasons:

(1) The quantitative information on edges is natural in real-life applications. For example, in online social networks, users' interaction frequency can be modeled as the edge weights in a graph. In collaboration networks, the number of cooperations between two scientists can be translated into the edge weights as well.
(2) Community search problem on graphs with nodes' features can be converted to community search problem on edge-weighted graphs. If the semantic meaning of nodes' features can represent the connectivity between two nodes, the similarity between features of two nodes can be modeled as the weight of the edge between them.

The key idea in tackling the community search problem in edge-weighted graphs is to find out a meaningful connectivity measurement and formulate an optimization problem on the searched community. [17, 20] aims to minimize the sum of all edge weights inside the community for evaluation and [9, 21] minimizes the maximal edge weight in a subgraph. Noted that in this report, we define the weights of edges as a measurement for distances between two nodes: the larger the weight is, the larger the weight of the edge is. This is different from some of past works. In some previous papers, the edge weight may also stand for the similarity score of two nodes, which means the weight is larger when two nodes are more similar. The details of these solutions will be further covered in *Section 2*.

However, the existing settings have the following limitations. Although [17, 20] make attempt to minimize the sum of all edge weights within the subgraph, such setting implicitly makes the size of queried community as small as possible. For [9, 21], they both adopt the maximal edge weight within a subgraph as measurement. Though it does not have any requirement on the subgraph size but the constraint posed on cohesiveness is too tight. How to allow relaxations is our major consideration.

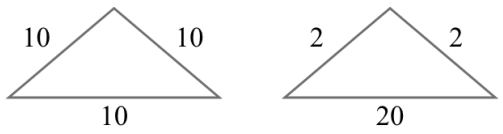


Figure 1: An example of motivating triangles: The left triangle has edge weights of 10,10,10 and the right one has edge weights of 2,2,20; in this case, the triangle weight threshold is set as 25.

The above limitations motivate us to propose our innovative idea on k-truss based community search in weighted graphs. Instead of using edge weights for evaluation, we now focus on the triangle weight (the sum of three edge weight in a triangle). Since triangle is the simplest stable structure unit to form a community, more relaxations are allowed. Instead of using one single edge to determine the connectivity of the whole community, using triangle to measure connectivity allows flexibility for a single edge as long as the triangles are considered as closely connected.

In Figure 1, there are two possible triangles. If we only look at the maximum of the edge weights, the left triangle (maximum weight: 10) is considered more closely connected than the right one (maximum weight: 20). However, in the right triangle, except for the edge of weight 20, the other two edge has relatively small weight 2, which reduces the overall triangle weight. One heuristic in social network analysis is that if two nodes both have closely connected edges to one node, then the edge between the two nodes is especially likely to become stronger [7]. In this sense, the right triangle is supposed to be preserved, although there exists a edge of weight 20, the two edges of weight 2 makes the three nodes potentially form a closely connected triangle. When we look at the left triangle, though every edge of weight 10 is much smaller than the maximum edge weight of the right triangle, but the overall triangle weight is 30, which is even larger than the right one (triangle weight: 24). Our purpose is to keep the right triangle but remove the left one. In addition, past works also use the sum of edge weights to measure the connectivity of a community. Nevertheless, this measurement limits the size of the searched result. Hence, using the average of edge weights will overcome this limitation. Another motivation of using triangle weight is that it relates to the average of edge weights in a community.

To further guarantee the connectivity of searched community, we require the user to input a triangle weight threshold. The threshold will determine whether the triangle is valid or not (if the triangle is valid, it means the triangle weight is no larger than the input threshold). The assumption behind is that the larger the edge weight, the looser the two nodes connect. In the previous example, the triangle threshold is set as 25 to preserve the right triangle and to remove the left one. Then we decompose and search communities on the valid subgraph.

Based on the motivations above, we propose our innovative solutions called weighted k-truss community search. By fixing an triangle weight threshold w and looking into all valid triangles with the sum of three edge weight below the threshold, we search the communities in edge-weighted graphs.

The organization of this paper is as follows. We discuss related works and analyze some limitations of previous works in *Section 2*. We provide our problem formulation in *Section 3*. In *Section 4* and *Section 5*, we introduce a top-down algorithm and a bottom-up algorithm respectively. Experiment results are reported in *Section 6*. Finally, we discuss and conclude our work in *Section 7*.

2 RELATED WORK

In the literature, there are several studies investigating into the community search problem on attributed graphs. These attributed graphs include three major classes: keyword-based attributed graphs, edge-weighted graphs, node-weighted graphs (a.k.a. influential graphs). Based on different properties of varied attributed graphs, existing works have proposed different solutions for searching the communities. Table 1 summarizes six related works and their proposed solutions in the order of publishing time.

In previous works, one important idea is to build index beforehand, taking up more space to store the structural information of the graph and speed up the query process [8, 11, 17, 21].

We can also conclude that existing methods can be summarized into two types: one is global search (search in a top-down manner) and the other one is local search (search in a bottom-up manner). Global search [8, 11, 20, 21] deletes unqualified nodes iteratively while local search [9, 17] starts from query nodes, expands to a subgraph and then search for the qualified communities. Generally, in regard with efficiency, global search is usually more time-consuming than local query algorithm. Although local search is faster, it cannot always give us optimal query results because the algorithm terminates as soon as it finds out any community that satisfies the required conditions. Following these two typical approaching methods, we also design two algorithms in this paper, one in top-down manner, and the other one in bottom-up manner.

In the remaining part, we then discuss several popular studies on keyword-based and edge-weighted graphs in details to illustrate the above ideas.

Method	Model	Node	Edge	Local Search	Index
[8]	k-core	keyword	×	×	✓
[11]	k-truss	keyword	×	×	✓
[21]	k-truss	×	weighted	×	✓
[20]	k-core	×	weighted	×	×
[9]	k-truss	×	weighted	✓	×
[17]	k-core	×	weighted	✓	✓

Table 1: A Comparison of Existing Solutions

2.1 Keyword-based

[8] and [11] aim to solve community search problem on keyword-attributed graphs, with k-core and k-truss respectively. [8] designs an innovative tree-based index, with the motivation that core numbers and keyword information are both nested. The index stores all the structural information and keywords to accelerate the community querying process. [11] also focuses on unweighted graphs with keyword attributes attached to all nodes. [11] proposes several attribute score functions to measure attribute coverage and correlation. The desirable community is required to cover as many query attributes as possible. Therefore, the intuition behind the score function is that more query attributes covered by the resulting subgraph, the higher the objective score; also more attributes shared by vertices of the resulting subgraph, the higher the score [11]. Along with the analysis of score function's properties, top-down greedy algorithm and index-based search algorithm are then proposed to conduct the query search.

2.2 Edge-weighted

[17, 20] propose solutions to intimate-core search problem in edge-weighted graphs. Existing works define *group weight* as the sum of all edge weights within a subgraph. Intimate-core search problem aim to find the community containing query nodes with the smallest *group weight*, satisfying core number requirement at the same time. The problem accepts multiple query nodes. [20] provides a global search method, which deletes unqualified node from the whole original graph. [17] offers a local graph search algorithm, involving connecting all the query nodes with a minimum spanning tree, expanding the tree to a local subgraph and making adjustment reducing the subgraph to a qualified community.

Apart from k-core structure, edge-weighted problem is also studied with k-truss structure [9, 21]. The goal of the paper is to find the top-r weighted k-truss communities, which is related to community detection rather than community search problem. [21] defines the minimum weight of edges in the subgraph as the connectivity measurement. The rationale behind this definition lies on the fact that the minimum weight value can guarantee the lower bound, i.e. each edge in the graph has a weight no smaller than it. Using extreme value as the threshold is able to provide strong robustness to outliers, which is regarded as the main advantage of such definition. However, the limitation is also obvious. Using minimum edge weight poses too much constraint on the community.

3 PRELIMINARIES

We consider a weighted but undirected graph $G = (V, E, W)$ with $n = |V|$ vertices and $m = |E|$ edges. W is the set of edge weights corresponding to each edge in E. Given such a graph $G = (V, E, W)$, a query vertex $v_q \in V$ and two integers k and w, our goal is to find all potential communities containing v_q based on modified k-truss model. Table 2 summarizes the notations used in this paper.

$G = (V, E)$	A graph with vertex set V and edge set E.
$G[H]$	The subgraph of G induced by a set of vertex H.
$deg_G(v)$	The degree of vertex v in G.
$N(v)$	The set of neighbors of a vertex v.
d_{max}	The maximum vertex degree in G.
$w(\triangle)$	Triangle weight.
▲	Valid triangle.
$sup(e, G)$	The support of an edge e in G.
$wsup(e, G)$	The weighted support of an edge e in G.
$\tau_w(H)$	Weighted trussness of a subgraph H.
$\tau_w(e)$	Weighted trussness of an edg e.

Table 2: Notations

We then introduce several definitions to formulate modified k-truss concept based on weighted graph. *Weighted k-truss community* is defined on the basis of *valid triangle* and *weighted support*.

Definition 3.1 (Typical Triangle). A typical triangle in G is a cycle of length 3. Let $x, y, z \in V$ be the three vertices on the cycle, and we denote typical triangle as $\triangle_{xyz}$.

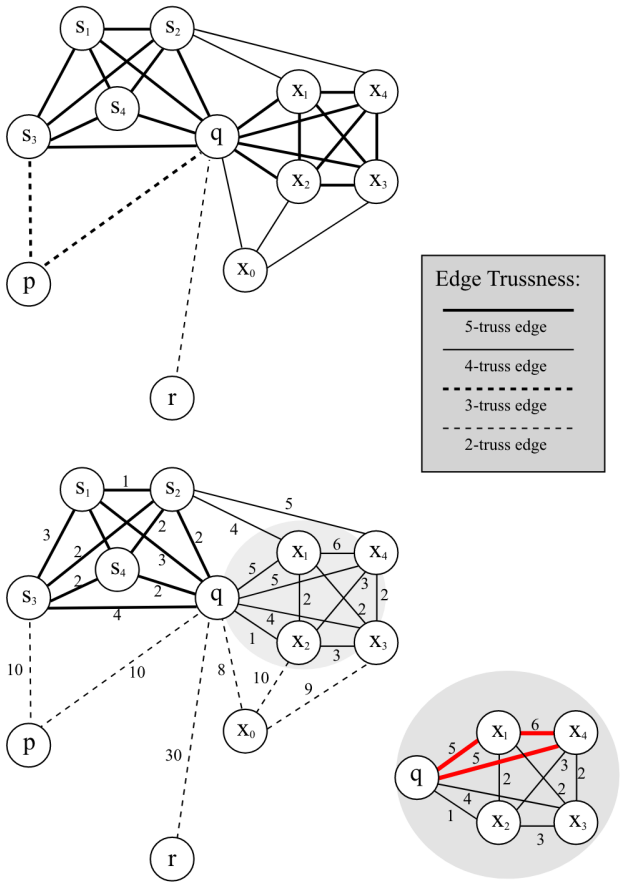


Figure 2: An illustrating example. The above figure shows edge trussness of unweighted graph; the below figure shows edge trussness of same graph with edge weights. Different levels of edge trussness is labeled in the gray box. The figure in the lower right is the zoomed part of the original graph with gray background. For the below figure, the weight threshold w is set as 15.

Definition 3.2 (Triangle Weight). The weight of $\triangle_{xyz}$ is defined as the sum of its three edges. We denote $w(\triangle_{xyz}) = w(x, y) + w(x, z) + w(y, z)$.

Definition 3.3 (Valid Triangle). A valid triangle in G is also a cycle of length 3. Let $x, y, z \in V$ be the three vertices on the cycle, and we denote valid triangle as $\blacktriangle_{xyz}$ only if $w(\triangle_{xyz}) \leq w$. Here w is a user-input threshold for valid triangle determination.

Definition 3.4 (Support). The support of an edge $e(x, y) \in E$ in G, denoted by $sup(e, G)$, is defined as $|\{\triangle_{xyz} : z \in V\}|$. In other word, it is the number of typical triangles that $e(x, y)$ can form with another node in G. When the context is obvious, we replace $sup(e, G)$ by $sup(e)$.

Definition 3.5 (Weighted Support). The weighted support of an edge $e(x, y) \in E$ in G, denoted by $wsup(e, G)$, is defined as $|\{\blacktriangle_{xyz} : z \in V\}|$. In other word, it is the number of valid triangles that $e(x, y)$ can form with another node in G. When the context is obvious, we replace $wsup(e, G)$ by $wsup(e)$.

Definition 3.6 (Weighted K-Truss Community). Given a graph G and an integer $k \geq 2$, G' is a weighted k-truss community, if G' satisfies the following three conditions:

(1) Weighted K-Truss. G' is a subgraph of G, denoted as $G' \subseteq G$, such that $\forall e \in E(G')$, $wsup(e, G') \geq k-2$;

(2) Edge Connectivity. $\forall e_1, e_2 \in E(G')$, $\exists \blacktriangle_1, \blacktriangle_2$ in G' such that $e_1 \in \blacktriangle_1$, $e_2 \in \blacktriangle_2$, then either $\blacktriangle_1 = \blacktriangle_2$, or $\blacktriangle_1$ is triangle connected with $\blacktriangle_2$ in G';

(3) Maximal Subgraph. G' is a maximal subgraph satisfying conditions (1) and (2). That is, $\nexists G'' \subseteq G$, such that $G' \subset G''$, and G'' satisfies conditions (1) and (2).

Definition 3.7 (Subgraph Weighted Trussness). The weighted trussness of a subgraph $H \subseteq G$ is the minimum weighted support of an edge in H, denoted by $\tau_w(H) = \min\{wsup(e, H) : e \in E(H)\}$.

Definition 3.8 (Edge Weighted Trussness). The weighted trussness of an edge $e \in E(G)$ is defined as $\tau_w(e) = \max_{H \subseteq G}\{\tau_w(H) : e \in E(H)\}$.

Example: To illustrate the definitions more clearly, we take the graph in Figure 2 as a concrete example. In this case, weight threshold w is set as 15. In the graph, $\triangle_{qx_1x_4}$ is a typical triangle with triangle weight 16. However, it is not a valid triangle since $w(\triangle_{qx_1x_4}) > 15$. As we can observe in the two graphs, the original unweighted subgraph $\{q, x_1, x_2, x_3, x_4\}$ is a 5-truss community, but in the weighted one, the subgraph $\{q, x_1, x_2, x_3, x_4\}$ becomes 4-truss. The difference results from the invalid triangle $\triangle_{qx_1x_4}$. Because of $\triangle_{qx_1x_4}$, the weighted support $wsup((q, x_1))$ and $wsup((q, x_4))$ are no longer 3 any more. The decrease in weighted support gives rise to the decrease in edge trussness, which makes sense because the community $\{q, x_1, x_2, x_3, x_4\}$ is not as tightly connected as the community $\{q, s_1, s_2, s_3, s_4\}$, given 15 as weight threshold.

4 GLOBAL QUERY

In this section, we firstly introduce a basic top-down decomposition method for weighted graphs, which is modified from truss decomposition algorithm in [10]. This top-down decomposition helps us obtain a weighted k-truss index based on weighted support. On the top of weighted k-truss index, then weighted k-truss community query can be conducted. The algorithm for query processing is borrowed from [10].

4.1 Weighted K-Truss Index

First of all, we introduce a weighted k-truss index, which is constructed by Algorithm 1 in a global manner. This algorithm requires an user input k and weight threshold w as well. To be specific, w as threshold is used to determine whether the triangle is valid when counting the support number.

Weighted K-Truss Index Construction. Weighted truss decomposition helps us obtain trussness $\tau_w(e)$ for each edge e in weighted graphs. As shown in Algorithm 1, we first initialize k as 2. As preparation of the algorithm, the weighted support for each edge is counted beforehand (the number of $wsup(e)$ increases by 1 only when the relevant triangle is valid). Then we sort all the edges according to their weighted support numbers in an ascending order. The essential step of the algorithm is that it iteratively removes a lowest weighted support edge $e(u, v)$ if $wsup(e) \leq k-2$. At the same time, for those who form a triangle with e and the sum of edge weights of the formed triangle is no larger than the threshold w, we also decrease their weighted support by 1 and reorder them after adjustment. Afterwards, the trussness k is assigned to the removed edge e. The terminating condition for this iteration is when all edges e with $wsup(e) \leq k-2$ are removed. Therefore, by this method, we finally decompose the weighted graph and obtain the trussness $\tau_w(e)$ of all edges e.

Algorithm 1: Weighted Truss Decomposition

```
Input: G = (V, E), weight threshold w
Output: τ(e) for each e ∈ E
1  k ← 2;
2  compute wsup(e) for each edge e ∈ E;
3  sort all the edges in ascending order of their weighted
     support;
4  while ∃e such that wsup(e) ⩽ (k − 2) do
5      let e = (u, v) be the edge with the lowest weighted
         support;
6      assume, w.l.o.g, deg(u) ⩽ deg(v);
7      for each w ∈ N(u) and (v, w) ∈ E do
8          if w(△_uvw) ≤ w then
9              sup((u, w)) ← wsup((u, w)) − 1;
10             sup((v, w)) ← wsup((v, w)) − 1;
11             reorder (u, w) and (v, w) according to their new
                 weighted support;
12     τ(e) ← k, remove e from G;
13 if not all edges in G are removed then
14     k ← k + 1;
15     goto Step 4;
16 return {τ(e)|e ∈ E};
```

Figure 2 shows the final decomposition results by indicating the trussness of each edge, the top one for unweighted graph and the bottom one for edge-weighted graph.

4.2 Weighted K-Truss Community Search

After the construction of weighted k-truss index, query algorithm 2 can then be performed to find weighted k-truss community. Since the weighted index tree structure is the same as the basic one in [10], we directly borrow the query algorithm to search for all potential k-truss communities. However, the index has already carry the information of edge weights, the result obtained will implicitly satisfy the weight threshold constraint.

Query Processing. As stated in [10], Algorithm 2 illustrates how to conduct k-truss community query based on the weighted index. The input of the algorithm is an integer k and a query node v_q, then it is expected to return us searched weighted k-truss communities containing v_q. To begin with, the algorithm looks into each incident edge (v_q, u) to find those with $\tau((v_q, u)) \geq k$ and push them into a queue Q one by one. The rationale behind is that such edge (v_q, u) can potentially be the seed edge to form a new truss community C_l [10]. Then *BFS* traversal search is performed to expand this community C_l with satisfying conditions. The formation of C_l will not terminate until Q becomes empty. Afterwards,

Algorithm 2: Query Process on Weighted K-Truss Index

Input: $G = (V, E)$, an integer k, query vertex v_q
Output: k-truss communities containing v_q

```
1  visited ← ∅; l ← 0;
2  for u ∈ N(v_q) do
3      if τ((v_q, u)) ≥ k and (v_q, u) ∉ visited then
4          l ← l + 1; C_l ← ∅; Q ← ∅;
5          Q.push((v_q, u)); visited ← visited ∪ {(v_q, u)};
6          while Q ≠ ∅ do
7              (x, y) ← Q.pop(); C_l ← C_l ∪ {(x, y)};
8              for z ∈ N(x) ∩ N(y) do
9                  if τ((x, z)) ≥ k and τ((y, z)) ≥ k then
10                     if (x, z) ∉ visited then
11                         Q.push((x, z));
                           visited ← visited ∪ {(x, z)};
12                     if (y, z) ∉ visited then
13                         Q.push((y, z));
                           visited ← visited ∪ {(y, z)};
14 return {C_1, ..., C_l};
```

the same process is performed to check next potential seed edge for a new community C_{l+1} until we obtain all remaining k-truss communities.

Example: Here we provide a modified example from [10] to illustrate the querying process. Assume our purpose is to find 5-truss communities with q as querying node in Figure 2 (below one). Algorithm 2 first picks an edge with trussness 5 and connected to querying vertex q as seed edge. For instance, we add edge (q, s_1) into Q. The *BFS* expansion then starts with edge (q, s_1), iteratively searching for neighboring edges like (q, s_2) which also satisfy trussness condition. All of searched edges are inserted into C_1 which is generated by seed edge (q, s_1). The termination of *BFS* expansion means the end of formation of C_1 community. The algorithm then checks other unvisited seed edges to form more communities if there exist. If taking unweighted graph into consideration, another community C_2 containing $\{q, x_1, x_2, x_3, x_4\}$ is also expected to be found.

5 LOCAL SEARCH

One important limitation of the global search method is that the algorithm is slow when encountering large graph dataset. Since the decomposition procedure iteratively deletes all the unqualified nodes, it takes long when the result community is rather small while the original graph is large. This motivates us to introduce a local search method. If the problem is solved in a bottom-up manner, there is no need to visit the whole graph but a limited local region around the query nodes, which dramatically save the time to decompose the graph. The local search algorithm for edge-weighted graph is inspired by the algorithm in [17]. The framework of the local search method is expanding query nodes into a subgraph with adapted *BFS* algorithm, then perform Algorithm 1 and 2 to do k-truss decomposition and query search on the subgraph until it finds a legitimate community, as shown in 3.

Algorithm 3: Local Search

Input: $G = (V, E, w)$, an integer k, query vertex v_q, weight w
Output: a k-truss community containing v_q

```
1  i ← 1;
2  while no community is found do
3      Expand v_q into i-hop neighborhood with Algorithm 4;
4      Apply Algorithm 1 to i-hop neighborhood ;
5      return if Algorithm 2 finds a community within i-hop
         neighborhood;
6      i ← i + 1;
```

Expanding From Query Nodes To Subgraph Algorithm 4 is the algorithm aiming to find a local neighborhood around the query node, which is an adapted version of *bfs*. We firstly record whether the nodes have been visited. The path length between each node and the query vertex is also kept in an array called *distance* (line 1). Initially we push the query node v_q into queue (line 2). Line 3 to line 10 pop out a node v from the queue and if the distance between v and v_q is less than or equal to i, we then iterate through all the neighbors of node v when the neighbor has not been visited before. If finding an unvisited node, we update its distance correspondingly (line 10).

Algorithm 4: Expanding Subgraph

Input: $G = (V, E, w)$, an integer i, query vertex v_q
Output: i-hop neighborhood containing v_q

```
1  visited(V) ← false; distance(V) ← 0;
2  Q.push(v_q);
3  while Q is not ∅ do
4      v ← Q.pop();
5      if distance(v) <= i then
6          for N(v) is not visited do
7              z ← N(v);
8              visited(z) ← true;
9              Q.push(z);
10             distance(z) = distance(v) + 1;
11 return i-hop neighborhood;
```

Example: Suppose we find a 2-hop neighborhood with query node A. Initially all the nodes are unvisited and the distances between each node and A are recorded as 0. We push A into queue Q. A is firstly popped out when entering the while loop. It iterates through *B, C, D*, mark them as visited nodes, push them into queue Q and update their distances to 1. Then, B is popped out from Q and B' unvisited neighbors E and F will be marked as visited, updated to distance of 2 and pushed to Q. After *C, D* are popped out from Q, no new nodes are found. When E is popped out, G is recorded to have a distance of 3 and pushed to Q. Since we are finding a 2-hop neighborhood, the algorithm stops here since the distance between A and G is 3 and returns the 2-hop neighborhood $\{A, B, C, D, E, F\}$.

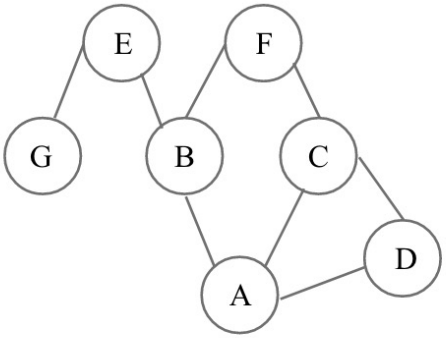


Figure 3: Expanding Subgraph Example

The idea behind algorithm 3 is also straightforward with this example. It decomposes and searches with algorithm 1 and 2 in the subgraph $\{A, B, C, D\}$, $\{A, B, C, D, E, F\}$, $\{A, B, C, D, E, F, G\}$ until one qualified community is discovered.

6 EXPERIMENTS

We present our experiment results in this section. *Section 6.1* shows the basic information of our dataset used for experiments. Then in *Section 6.2*, we discuss the evaluation results for our algorithms. We make comparison in *Section 6.3* regarding to their algorithm performance. Finally, we provide a case study for a deeper understanding of the performance of the algorithms in *Section 6.4*.

6.1 Dataset Overview

We use real-world weighted graph dataset in our experiments: DBLP (https://dblp.uni-trier.de/xml/). A vertex is an author and an edge represents the connection between two authors. In our setting, if two authors have co-authored a work once, there is an edge between them. The weight of an edge $((u, v))$ is defined as $\frac{100}{coop(u,v)}$, where $coop(u, v)$ denotes the number of times that the authors have cooperated. Table 3 shows the basic information of DBLP.

Dataset	Vertices	Edges	Average Edge Weight
DBLP	996674	3966007	86.7

Table 3: Dataset used in our experiments.

6.2 Evaluation

In this part, we provide several evaluation measurements on our proposed algorithms, comparing with basic k-truss algorithm. For convenience, we name the global algorithm in section 4 as $WTC-G$ and the local algorithm in section 5 as $WTC-L$, which refers to Weighted Truss Community - Global and Weighted Truss Community - Local respectively. Besides, to measure the performance and make comparison in a quantitative way, we provide two possible measurements.

(1) Average edge weight is the sum of all edge weights divided by the number of edge within a community. We denote average edge weight as $\overline{w_e}$. $\overline{w_e} = \frac{\sum w(e)}{|e|}$.
(2) Average triangle weight is the sum of all triangle weights divided by the number of triangle within a community. We denote average triangle weight as $\overline{w_\triangle}$, which is computed as $\frac{\sum w(\triangle)}{|\triangle|}$.

Weight distribution. To begin with our experiment analysis, we first conduct a specific case on the fixed setting. By choosing k=4 and w=200 (only for $WTC-G$ and $WTC-L$), communities are then searched on basic k-truss algorithm, $WTC-G$ and $WTC-L$ respectively. To illustrate the differences among three resulted communities, we compute the distribution of w_e and $w_\triangle$ for all communities, as shown in Figure 4 (a) (b). From our observation, the histograms of $WTC-G$ and $WTC-L$ are both in bell shape, with highest bar standing in the middle. However, for basic k-truss algorithm, the highest bar always stands in the right-most range, which means the majority of edges have higher weights. Since we have already set the weight threshold to be 200, most triangle weights stay within the range from 101 to 200. But there are still a few triangles have exceeding triangle weight because some edges form invalid triangles by coincidence and the removal of these triangles will not affect the community's overall trussness. To draw a conclusion, after posing constraint on triangle weights, the searched communities have better cohesiveness on coauthorship. Authors within the communities are much more tightly connected.

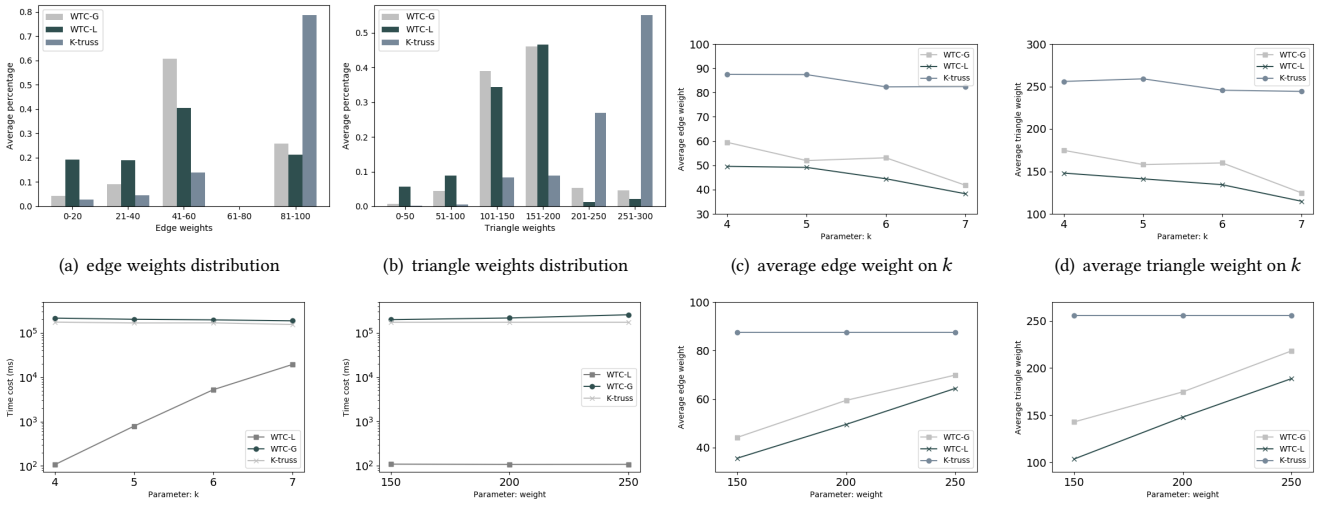
k sensitivity analysis. Figure 4 (c) (d) shows $\overline{w_e}$ and $\overline{w_\triangle}$ when parameter k is varied. The figures clearly illustrate the gap between the basic k-truss algorithm and our proposed algorithms: $\overline{w_e}$ and $\overline{w_\triangle}$ of basic k-truss are obviously higher than $WTC-G$ and $WTC-L$ no matter at whichever k. Such result is simply because of the posed constraint on weight threshold. And generally, when k is getting larger, the weight will also becomes smaller.

Another clear observation is that $WTC-L$ has smaller weights on varied k than $WTC-G$ algorithm. The reason behind is that $WTC-G$ outputs the maximal community but barely able to satisfy the user's requirement. However, $WTC-L$ only finds out the first solution and then terminate the algorithm. Based on such rational, $WTC-L$ gets better community than $WTC-G$ in terms of $\overline{w_e}$ and $\overline{w_\triangle}$.

w sensitivity analysis. Figure 4 (g) (h) shows $\overline{w_e}$ and $\overline{w_\triangle}$ when weight parameter w is varied. It is clearly seen that the result communities from $WTC-G$ and $WTC-L$ have smaller $\overline{w_e}$ and $\overline{w_\triangle}$. Hence, $WTC-G$ and $WTC-L$ are both effective in searching for densely connected communities. In addition, we can observe that weight w serves as an effective threshold to control the average triangle weights of the communities. Meanwhile, this reduces average edge weights of result communities.

Another observation is that for Figure 4 (g) (h), $WTC-L$ has smaller weights than $WTC-G$ for both edge weights and triangle weights. The reasons behind is similar to what we have mentioned in *k sensitivity analysis*.

Time complexity analysis. As is shown in Figure 4 (e) (f), we can observe that $WTC-G$ is slightly slower than basic k-truss algorithm, since $WTC-G$ adds triangle check steps to the basic k-truss algorithm. $WTC-L$ performs much more efficient, compared with



(a) edge weights distribution (b) triangle weights distribution (c) average edge weight on k (d) average triangle weight on k

(e) time complexity on k (f) time complexity on w (g) average edge weight on w (h) average triangle weight on w

Figure 4: Experiment results on k sensitivity and w sensitivity. Figure (a) (b) has fixed setting with k=4 and w=200.

$WTC-G$ and basic k-truss method. From Figure 4 (e), as k increases, $WTC-L$ becomes slower since the result community should be contained in a larger neighborhood. Searching in a larger neighborhood involves more iterations of Algorithm 3, thus causing the performance to be less efficient. When varying weight parameter w, time costs of $WTC-G$ and $WTC-L$ remain similar. We can conclude that when k is small, $WTC-L$ has a better performance in terms of efficiency.

6.3 Case Study on DBLP Network

We test our algorithms on the DBLP network to find potential communities. Figure 5 is a subgraph within DBLP dateset, which shows us a possible 4-truss community. The darkness of the edge between two authors represent the edge weight between them. Here, since the edge weight is the distance of two nodes, which means the darker the edge is, the closer the two nodes are. Hence, the edge is darker if the authors have a shorter distance (co-author more frequently). In basic k-truss algorithm, the whole subgraph is considered as a 4-truss community. However, we can clearly see that "Kevin W. Hamlen" is loosely connected with the neighbors because the weights of edges to other nodes are all 100. Our proposed solution intends to dismiss these loosely connected nodes. Therefore, the pentagon consisting of five authors {"Jiawei Han", "Xifeng Yan", "Jing Gao", "Philip S. Yu", "Jian Pei"} is a qualified 4-truss community with our query $k = 4, w = 200$. We can observe that the result community from our solution is more closely connected, given the weight threshold w. In basic k-truss algorithm, {"Kevin W. Hamlen", "Mohammad M. Masud", "Jing Gao", "Jiawei Han"} is contained in a 4-truss community. Although {"Mohammad M. Masud", "Jing Gao", "Jiawei Han"} is a valid triangle, {"Kevin W. Hamlen", "Mohammad M. Masud", "Jiawei Han"} is invalid with threshold $w = 200$. Hence, the weight support $wsup$ of edge between "Mohammad M. Masud" and "Jiawei Han" is only 1 and therefore {"Mohammad M. Masud", "Jing Gao", "Jiawei Han"} is not contained in 4-truss community by our setting.

Statistically speaking, the average edge weight $\overline{w_e}$ for the whole 4-truss community is 38.785 but $\overline{w_e}$ for the pentagon is 24.889. In terms of average triangle weight, $\overline{w_\triangle}$ for the whole graph is 113.182 and $\overline{w_\triangle}$ for the pentagon is 72.429. It is obvious that our setting is able to output us a more satisfying result community with smaller average edge weight and triangle weight.

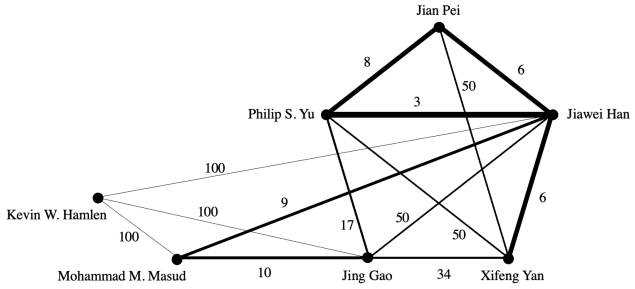


Figure 5: A case study on DBLP Network showing a possible 4-truss community. The smaller the edge weight, the deeper the link between two authors and the more connected coauthorship between them.

7 DISCUSSION & CONCLUSION

Inspired by the special triangle structure built in k-truss communities, we propose weighted truss community with a triangle weight threshold, which results in relatively better connectivity within community. This is justified by smaller average edge weights and triangle weights in our experiments.

Two algorithms, one in top-down manner and the other on bottom-up manner are also designed to perform community search task on edge-weighted graphs. In terms of effectiveness comparison, the global algorithm outputs all potential communities satisfying the user input requirement but the local one terminates as long as

it finds out first one inside the n-hop subgraph. The local algorithm does not gives us a global optimal solution.

Generally, local search algorithm' time cost is much smaller than global search algorithm. The local algorithm performs efficiently especially when k is small (the user hopes to obtain a small community). However, when k is large or the user queries a large community, local search algorithm goes through too many iterations to obtain the first community satisfying the requirement, causing it to be slow.

Future Extension. Both two algorithms are online search algorithms. For further improvement of efficiency, it is possible to use index-based solution to facilitate query search. For example, TCP-index can be considered to adjust to solving this problem, since it not only improves the efficiency and support dynamic index updating.

Another potential improvement is regarding the flexibility of input query nodes. Our solutions only support one single query node. [17] provides ideas about taking in multiple query nodes. Multiple query nodes can be spanned into one component with minimum spanning tree and then perform 3.

Additionally, another possible extension is on user input threshold, which can be better formulated. The edge weights in this paper are not normalized, so it is hard for user to choose a reasonable threshold as input requirement without having an understanding on the graph data. The better way to handle it is to ask user for a number ranging from 0 to 1 as threshold input.

In this paper, we propose a measurement called triangle weight, composed of weights of three edges in a triangle. Based on the heuristic that within a triangle, if two edges are closely connected, the third edge may potentially become closer, triangle weight measurement can be modified into relaxed triangle weight, which only considers weights of two edges in a triangle. This extension may give more relaxations to the problem.

Contribution. Both members in the group contribute to the final year project equally. Specifically speaking, during the paper reading phase, Chen Yu focuses more on k-core intimate community in attributed graphs, but Xu Xiang concentrates on k-truss algorithm researching. The innovative idea we implemented this term was firstly proposed by Xu Xiang. Global search algorithm was from our discussion and Yu proposed local search method. Both were developed and coded together. As for the experiment part, Xiang is responsible for running codes and recording the result; Yu is in charge of running codes and graph making.

Acknowledgement. Here we would like to express our great gratitude to our supervisor Cheng Hong and our tutor Yuli Jiang. Thanks for their kind guidance and suggestions all way along our research journey. Thanks to their help, we are able to push forward our project quickly and smoothly.

REFERENCES

[1] James Abello, Mauricio GC Resende, and Sandra Sudarsky. 2002. Massive quasi-clique detection. In *Latin American symposium on theoretical informatics*. Springer, 598–612.

[2] Yong-Yeol Ahn, James P Bagrow, and Sune Lehmann. 2010. Link communities reveal multiscale complexity in networks. *nature* 466, 7307 (2010), 761–764.

[3] Vladimir Batagelj and Andrej Mrvar. 1998. Pajek-program for large network analysis. *Connections* 21, 2 (1998), 47–57.

[4] Vladimir Batagelj and Matjaz Zaversnik. 2003. An O (m) algorithm for cores decomposition of networks. *arXiv preprint cs/0310049* (2003).

[5] Jonathan Cohen. 2008. Trusses: Cohesive subgraphs for social network analysis. *National security agency technical report* 16 (2008), 3–29.

[6] Wanyun Cui, Yanghua Xiao, Haixun Wang, and Wei Wang. 2014. Local search of communities in large graphs. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 991–1002.

[7] David Easley, Jon Kleinberg, et al. 2010. *Networks, crowds, and markets*. Vol. 8. Cambridge university press Cambridge.

[8] Yixiang Fang, Reynold Cheng, Siqiang Luo, and Jiafeng Hu. 2016. Effective community search for large attributed graphs. *Proceedings of the VLDB Endowment* 9, 12 (2016), 1233–1244.

[9] Wafaa MA Habib, Hoda MO Mokhtar, and Mohamed E El-Sharkawi. 2020. Weight-Based K-Truss Community Search via Edge Attachment. *IEEE Access* 8 (2020), 148841–148852.

[10] Xin Huang, Hong Cheng, Lu Qin, Wentao Tian, and Jeffrey Xu Yu. 2014. Querying k-truss community in large and dynamic graphs. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 1311–1322.

[11] Xin Huang and Laks VS Lakshmanan. 2017. Attribute-driven community search. *Proceedings of the VLDB Endowment* 10, 9 (2017), 949–960.

[12] Mark EJ Newman and Michelle Girvan. 2004. Finding and evaluating community structure in networks. *Physical review E* 69, 2 (2004), 026113.

[13] Gergely Palla, Imre Derényi, Illés Farkas, and Tamás Vicsek. 2005. Uncovering the overlapping community structure of complex networks in nature and society. *nature* 435, 7043 (2005), 814–818.

[14] Martin Rosvall and Carl T Bergstrom. 2008. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences* 105, 4 (2008), 1118–1123.

[15] Stephen B Seidman. 1983. Network structure and minimum degree. *Social networks* 5, 3 (1983), 269–287.

[16] Mauro Sozio and Aristides Gionis. 2010. The community-search problem and how to plan a successful cocktail party. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. 939–948.

[17] Longxu Sun, Xin Huang, Rong-Hua Li, and Jianliang Xu. 2020. Fast Algorithms for Intimate-Core Group Search in Weighted Graphs. In *International Conference on Web Information Systems Engineering*. Springer, 728–744.

[18] Jia Wang and James Cheng. 2012. Truss decomposition in massive networks. *arXiv preprint arXiv:1205.6693* (2012).

[19] Jierui Xie, Stephen Kelley, and Boleslaw K Szymanski. 2013. Overlapping community detection in networks: The state-of-the-art and comparative study. *Acm computing surveys (csur)* 45, 4 (2013), 1–35.

[20] Dong Zheng, Jianquan Liu, Rong-Hua Li, Cigdem Aslay, Yi-Cheng Chen, and Xin Huang. 2017. Querying intimate-core groups in weighted graphs. In *2017 IEEE 11th International Conference on Semantic Computing (ICSC)*. IEEE, 156–163.

[21] Zibin Zheng, Fanghua Ye, Rong-Hua Li, Guohui Ling, and Tan Jin. 2017. Finding weighted k-truss communities in large networks. *Information Sciences* 417 (2017), 344–360.