IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 3, 2020, accepted August 10, 2020, date of publication August 13, 2020, date of current version August 24, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3016214

Weight-Based K-Truss Community Search via Edge Attachment

WAFAA M. A. HABIB, HODA M. O. MOKHTAR, AND MOHAMED E. EL-SHARKAWI
Information Systems Department, Faculty of Computers and Artificial Intelligence, Cairo University, Cairo 12613, Egypt
Corresponding author: Wafaa M. A. Habib (w.momen@fci-cu.edu.eg)

ABSTRACT Community search is the task of discovering dense subgraph that satisfy a set of given query parameters. Most community search algorithms consider link structure while ignoring link weight. A recent study proposed the idea of discovering weighted communities which focuses on both link structure and link weight using an online search approach and index-based approach. In this paper two online algorithms are proposed to scale-up the existing online approach efficiency. Performance evaluation of the proposed algorithms against the existing online approach over different datasets shows a great improvement in terms of search and query evaluation time.

INDEX TERMS Community search, query processing.

I. INTRODUCTION

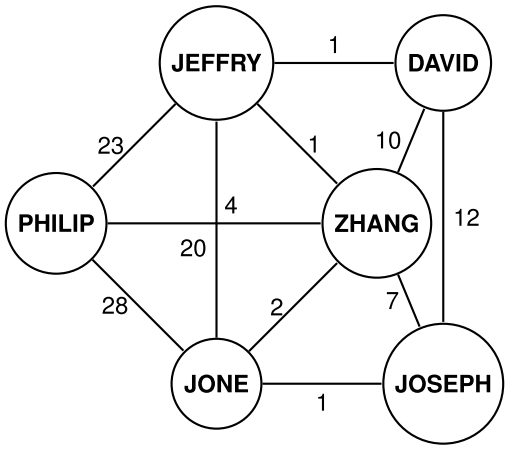
Due to the rapid growth of complex data, the graph model has been increasingly used to model various interactions between entities. There are many interesting applications such as social networks, biological networks, and citation networks that have been modeled as a graph. In such applications, community structure exists as a subgraph of strongly interconnected vertices [1]. One of the main task is to discover such community structure using community search approaches. Community search is the task of discovering communities that contain a given set of query nodes or satisfy certain query parameters. Community search with its variations have been a recent research direction which is studied extensively in literature especially on large networks [2]–[10]. However, the majority of those studies ignore other community aspects and mainly the edge weight. Edge weight is used to indicate the strength between any two nodes. Ignoring the edge weight causes the loss of important information within the discovered communities. Below are some examples of real networks where edge weight has a significant role:

- In the co-authorship networks, the edge weight may represent the number of papers the two linked authors had co-authored together in general or in a specific research area (*i.e.*, Machine Learning, Database, etc) [11]. Ignoring the edge weight within the discovered communities would conceal the strength of the co-authorship between different authors.
- In social networks, the edge weight may denote the similarity, or interactions between users [11]. Ignoring the edge weight would result in communities of dissimilar 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 [12]. Ignoring the edge weight would obscure and hide the business ties between countries.

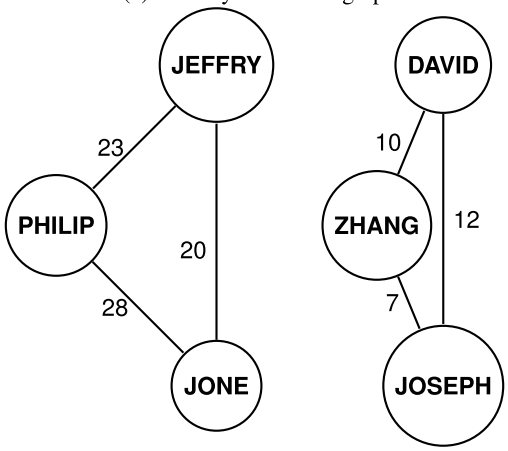
To illustrate the edge weight importance, consider the graph in FIGURE 1a where the graph represents an example of a collaboration network. Each vertex in the graph represents an author while edge weight represents the number of papers that linked authors had co-authored together. The graph in FIGURE 1a is a densely connected component given its structure. On the other hand, some authors mentioned in this graph are weakly connected in terms of co-authorship (i.e. JEFFRY and ZHANG had co-authored only one paper together). The two subgraphs in FIGURE 1b are densely connected given their structures. In addition, the high weight on the edges of each subgraph show a strong co-authorship between authors. For Example, in the two subgraphs the minimum number of papers that any linked authors co-authored is 20 and 7. The subgraph with minimum edge weight 20 is considered as the top weighted subgraph while the subgraph with minimum edge weight 7 is considered as the second weighted subgraph.

Motivated by the importance of edge weight, this paper focuses on discovering top weighted communities. More specifically, we are interested in discovering the *top-r*

The associate editor coordinating the review of this manuscript and approving it for publication was Huiling Chen.



(a) Densely connected graph.



(b) Densely connected subgraphs with strong authorship.

FIGURE 1. Motivation example.

weighted k-truss communities. k-truss community detection model is chosen to ensure cohesiveness by applying a constraint that each edge must be contained in at least k-2 triangles where each triangle models the relationship between three nodes.

Formally a weighted k-truss community in a graph G is a subgraph $H \subset G$ where edges in H have a minimum number of k-2 common triangles between them. In addition, H is the maximal subgraph with the highest weight.

A. EXISTING APPROACHES AND THEIR DRAWBACKS

The Existing approaches to discover *top-r weighted k-truss* communities can be classified as an online search algorithm and an index-based search algorithm.

Online Search Algorithms [11]: The online search algorithms compute the communities based on the k-truss decomposition of the entire graph. The edges with the minimum weight are iteratively removed, and with each removal the maximal connected component procedure is run to find the next connected component. Nevertheless, the online search algorithm can't scale for large graphs as it has to run the maximal connected component procedure multiple times.

Index-based Algorithms [11]: The index-based algorithms are used to efficiently retrieve the *top-r weighted k-truss* communities from a pre-built index that stores all the weighted k-truss communities for each k in the main memory. The index strategy has improved the query processing time while discovering communities; however, the index size has increased enormously. Besides, it would be time-consuming to update the index with such enormous index size.

B. OUR MAIN CONTRIBUTION IN THIS PAPER

The goal of this work is to propose two new algorithms, the `BACKWARD ALGORITHM`, and `WEIGHT-SENSITIVE LOCAL SEARCH ALGORITHM (WSLSA)` to overcome the drawbacks of the existing algorithms, more specifically the online search algorithm [11]. The concept of local search is employing in developing the two proposed algorithms for discovering *top-r* weighted communities. 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. More specifically, our contributions are as follows:

- A `BACKWARD ALGORITHM` is proposed for the fast discovery of the *top-r weighted* communities. This algorithm performs efficiently to detect communities by visiting edges with the highest weight and satisfying at least a certain level of trussness.
- A `WEIGHT-SENSITIVE LOCAL SEARCH ALGORITHM (WSLSA)` is proposed to detect communities by visiting edges with the highest weight regardless of their trussness level. The algorithm time complexity is linearly proportional to the number of edges visited to detect the *top-r* weighted communities.
- Extensive experiments are conducted on different graphs with different sizes. Experiment results prove the efficiency of the proposed algorithms.

The rest of this paper is organized as follows: Section II presents related work. Section III overviews some of the basic concepts used in the paper including weighted graph, edge support, and weighted k-truss communities. In section IV the proposed algorithms are presented. Section V presents the empirical results and discusses them. Finally, Section VI concludes the paper and highlights possible directions for future work.

II. RELATED WORK

Extracting communities from the graph is classified as either the task of community detection or community search.

Community detection is well studied in literature [13]–[16], and is defined as the task of discovering a group of nodes that are highly intra-connected to each other and weakly connected to outside vertices. In [17], [18] the authors

proposed different techniques to divide the graph into a set of partitions with a predefined number of nodes such that the number of inter-edges is minimized.

Different models have been studied in literature to discover dense subgraphs such as cliques [19], [20], quasi-clique [21], k-core [22], [23], edge density [24], [25], edge connectivity [26], [27], and k-truss [2], [3]. A *k-core* is a subgraph in which each node has at least degree k. A *k-truss* is a subgraph in which each edge must be contained in at least *k-2* triangles, *k-truss* is also (*k*-1)-core but not vice-versa; *k-truss* is a *k*-1 edge connected. Both *k-core* and *k-truss* models can be computed in linear time regardless of graph size. Authors in [28] proposed a new model called KTMiner to detect k-truss communities in a disturbed manner using Map-Reduce framework on Apache Spark environment.

Another direction of community detection is the community search which focuses on finding a subgraph that is cohesive and contains a given query vertex or a set of query vertices. The community search problem was well studied in literature [5]–[7], [29]–[31]. In [29] the authors proposed a global search technique to find a community containing a query vertex in linear time by iteratively removing vertices with the minimum degree. In [5] the authors proposed a more efficient local search technique for the same problem. A novel α-adjacency γ-quasi-k-clique model was proposed by the authors in [30] to study the overlapping community search problem. In [6], [31], the *k-truss* model is utilized to study the community search problem, where the maximal connected *k-truss* component containing a query vertex is considered as a community. However, all this work does not consider the weight of the community. A community search in a node-weighted network -influential community search- was studied first by the authors in [4]. The authors in [4] proposed two techniques online based and index based to find *top-r* most influential communities based on *k-core* model. Influential community search is also recently studied in [9], [32], the authors in [9] proposed two useful extensions to the online search algorithm proposed in [4], namely forward and backward algorithms. The backward algorithm starts with the most influential node and builds communities from the most important one to the least important one unlike the forward algorithm. An efficient local search algorithm was presented by the authors in [32] which is considered as the state of the art. It starts by minimizing the size of the original graph to a subgraph having only nodes with influence greater than a threshold. All these techniques are node-weight based which utilize the *k-core* model as their cohesion measure.

III. PRELIMINARIES

In this section we discuss some of the basic concepts that will be used in the rest of this paper. In this paper, an undirected and edge-weighted graph is denoted as $G = (V, E, W)$, where V is the set of vertices,E is the set of edges and W is the set of edge weights where each edge $e(u, \upsilon) \in \mathrm{E}$ is assigned a weight $\omega(e)$. The set of neighbors of each 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 graph $\mathrm{H} = (\mathrm{V_H}, \mathrm{E_H}, \mathrm{W_H})$ is called an induced subgraph of G iff: $\mathrm{V_H} \subseteq \mathrm{V}$, and $\mathrm{E_H} = \{(u, \upsilon)$: $u,\upsilon \in \mathrm{V_H} \wedge (u,\upsilon) \in \mathrm{E}\ \}$. The weight of H is defined as the minimum weight of edges since the minimum weight would ensure that all the edges in *H* have at least the minimum weight. In addition, the minimum weight would be robust to outliers unlike using the average of edge weights which is sensitive to outliers as discussed in [4].

Definition 1 (Community Weight): Given a subgraph $\mathrm{H} = (\mathrm{V_H}, \mathrm{E_H}, \mathrm{W_H})$, the weight value of H denoted by $f(\mathrm{H})$, is defined as the minimum weight of the edges, i.e., $f(\mathrm{H}) = \min_{e\in \mathrm{E_H}}\{\omega(e)\}$

For cohesiveness measure to discover dense subgraph, there are many algorithms that have been proposed like k-core [33], [34], k-truss [2], [3], edge-connectivity [26], [27], and clique or quasi-clique [21], [30].

In this paper, we consider k-truss for our community search model as it leads to strongly connected communities compared to other algorithms. *k-truss* model deduces the cohesion of a community through counting the number of triangles for each edge in the graph.

A triangle is the concept used to describe a cycle of connections between 3 nodes in the graph G. A triangle involving vertices u,υ,w, is denoted by $\triangle_{u\upsilon w}$. Then the support of an edge $(u,\upsilon) \in \mathrm{E}$ in H is defined as follows:

Definition 2 (Edge Support): The support of an edge $e(u, \upsilon) \in \mathrm{E_H}$, denoted as sup(*e*,H), is the number of triangles containing *e*, *i.e.*, $\mathrm{sup}(e,\mathrm{H}) = \{|\triangle_{u\upsilon w}|: w \in \mathrm{V_H}\}$

Based on the definition of Edge Support and triangle, we define the connected k-truss community as follow:

Definition 3 (k-Truss Community): Given a subgraph $\mathrm{H} = (\mathrm{V_H}, \mathrm{E_H}, \mathrm{W_H}) \subseteq \mathrm{G}$, and $\mathrm{k} \geq 2$, H is a k-truss community, if it satisfies the following two conditions:

- **k-Truss**: for each edge $\mathrm{e}(u,\upsilon) \in \mathrm{E_H}$, $\mathrm{sup}(e,\mathrm{H}) \geq (\mathrm{k}-2)$;
- **Maximal Subgraph**: there exists no other k-truss H' of G such that $\mathrm{H} \subset \mathrm{H}'$.

The definition of k-truss community implies another two definitions for community trussness and edge trussness.

Definition 4 (Community Trussness): The trussness of a community $\mathrm{H} \subseteq \mathrm{G}$, denoted as $\tau(\mathrm{H})$, is the minimum support of edges in H, such that $\tau(\mathrm{H}) = \min\{\mathrm{sup}(e, \mathrm{H}): e \in \mathrm{E_H}\}$.

By this definition, the edge can reside in multiple subgraphs for different k values. Truss number for an edge is defined to be the largest k value for which there is a k-truss community that contains the edge.

Definition 5 (Edge Trussness): The trussness of an edge $e(u,\upsilon) \in \mathrm{E(G)}$, denoted as $\tau(\mathrm{e})$, is the trussness of the maximal subgraph containing *e*, such that $\tau(\mathrm{e}) = \max_{\mathrm{H}\subseteq\mathrm{G}}\{\tau(\mathrm{H}): e \in \mathrm{E_H}\}$

Based on the previous definitions of community weight, and k-truss community, we define the weighted k-truss community as follow:

Definition 6 (Weighted K-Truss Community): Given an undirected and edge-weighted graph $G = (V, E, W)$ and an

integer k, a Weighted K-Truss Community is an induced subgraph H of G that satisfies the following:

- **Connectivity:** H is a connected subgraph;
- **Cohesiveness:** Each edge e in H has minimum support at least $k-2$ triangles;
- **Maximal:** H is a maximal induced subgraph that is a connected, cohesiveness, and there exists no other induced subgraph H′ of G satisfying conditions(connectivity and cohesiveness), and additionally $\neg\exists$: $H \subset H' \wedge f(H')=f(H)$.

Since the minimum weight of the k-truss community is used as the community weight, then the edge with minimum weight is used as a key-edge for the k-truss community. key-edge is defined as below.

Definition 7: An edge e is a key-edge regarding k value, if there is a subgraph H such that the weight of H $f(H) = w(e)$ and the minimum support of edges in H is at least $k-2$.

By applying cohesiveness and connectivity constraints during extracting a community, the resulting community is guaranteed to be a k-truss community. And by applying the maximal constraint the resulting weighted k-truss community is guaranteed not to be a subset from another weighted k-truss community with the same weight.

Example 1: Consider the graph in FIGURE 2a. Suppose for instance k = 3*, as clarified in Definition 6 the graph itself is a weighted 3-truss community with weight value* = 3*. Also, there are two weighted communities; the 3-truss subgraph shown in FIGURE 6c with weight value 18, and 4-truss subgraph shown in FIGURE 2c with weight value 15. The subgraph shown in FIGURE 2d also has weight 15; however it is not weighted* 3 − *truss community as it is contained in the subgraph shown in FIGURE 2c with the same weight 15.*

Problem Statement: Given an undirected and edge-weighted graph G = (V, E, W)*, and two query parameters r and k, the problem of* weighted k-truss community *is to discover the* top-r weighted k-truss *community.*

Example 2: Consider the example illustrated in FIGURE 2a, suppose r = 3 and k = 3, the top-3 weighted 3-truss communities, denoted as $H_{k,r}$, are, $H_{3,1}$ has shown in FIGURE 6c with weight value 18, $H_{3,2}$ shown in FIGURE 2c, with weight value 15, and $H_{3,3}$ shown in FIGURE 2e with weight value 8.

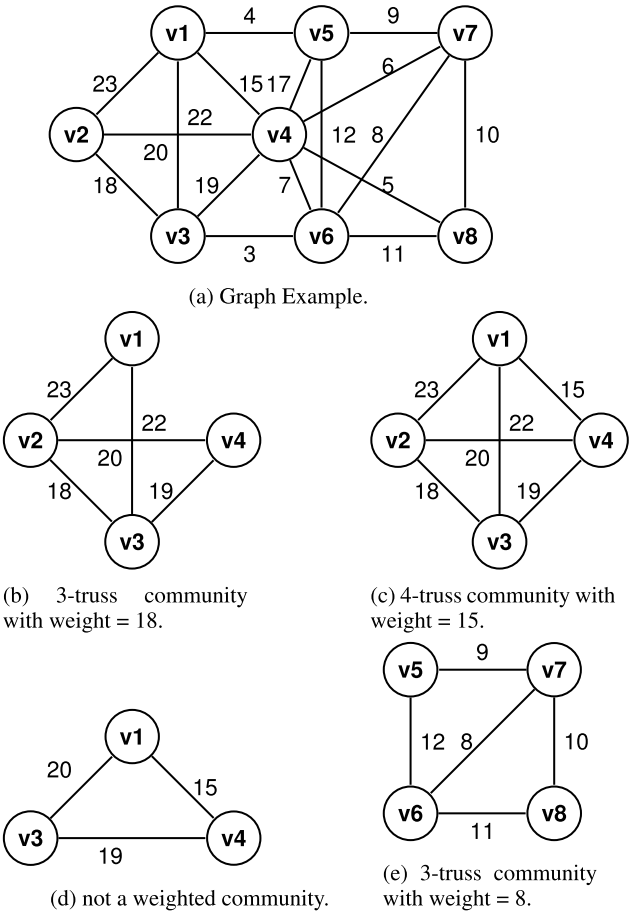


FIGURE 2. Graph example and its weighted communities.

IV. PROPOSED ALGORITHMS

In this section the proposed algorithms to discover all the *top-r weighted k-truss* communities are discussed. The concept of local search is utilized to find communities from the highest weight to the smallest one. First the online search `BACKWARD ALGORITHM` is introduced to find *top-r weighted k-truss* efficiently. Then, a `WEIGHT-SENSITIVE LOCAL SEARCH ALGORITHM (WSLSA)` is proposed to accelerate the `BACKWARD` one and efficiently discover all top-r weighted k-truss communities. Before proceeding further, two important lemmas are introduced:

Lemma 1: Upon the insertion of an edge $e(u, \upsilon)$ in subgraph H, if a triangle of $\triangle_{uvw}$ is formed, then the support of the affected edges $e'(u, w)$, and $e''(\upsilon,w)$ increases by 1.

Proof by Contradiction: By removing an edge e, the support of the affected edge e′ will be reduced by only one, as e and e′ exist in only one triangle. Using edge removal effect, the insertion of an edge effect can be proved by contradiction. Suppose inserting as edge e increases the support of the affected e′ edge by more than 1, then deleting an edge reduces support by more than one as well. This contradicts with deletion.

Lemma 2: Upon the insertion of an edge $e(u, \upsilon)$ in subgraph H, if H is found to be a connected k-truss community, then $e(u, \upsilon)$ support in H has to be at least $k-2$. In other words, $e(u, \upsilon)$ exists in at least $k-2$triangles with $k-2$ different nodes. Consequently, $e(u, \upsilon)$ is connected to $2*(k-2)$ affected edges having support at least $k-2$.

Proof: By definition, any k-truss connected component must have at least k nodes with a degree at least k-1.

A. BACKWARD ALGORITHM

The `BACKWARD ALGORITHM` initially starts by finding the maximal *k-truss* H of G. Based on H, communities are built incrementally, a candidate subgraph is generated by attaching

edges with the highest weight. The generated candidate has to be verified to confirm the existence of a k-truss community.

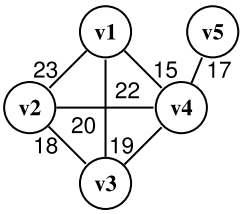
Algorithm 1 shows the pseudo-code of this process.

Algorithm 1 *BACKWARD ALGORITHM*

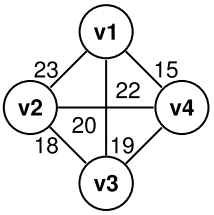
Input: An undirected graph G = (V, E, ω) and two parameters r, k

Output: The top-r weighted k-truss communities

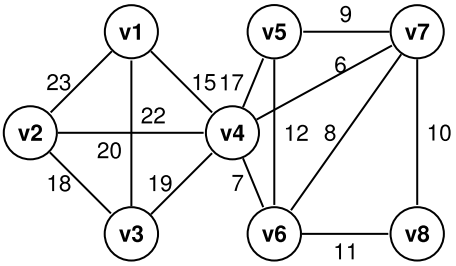
1: Find The Maximal K-Truss H of G
2: **for each** $e \in E_H$ **do**
3: Initialize $disconnected[e] = true$
4: Initialize $support[e] = 0$
5: **end for**
6: i=1
7: minsup=0
8: Initialize $Graph\_Instance(G') = \emptyset$
9: **repeat**
10: Find $e(u,\upsilon) \in E_H$ with the highest weight edge
11: Move $e(u,\upsilon)$ from H to G′
12: **if** $(nb(u) \cap nb(\upsilon) \neq \emptyset)$ **then**
13: Flag1 = UpdateSupport(G', $e(u, \upsilon)$)
14: **end if**
15: **if** (Flag1 == True) **then**
16: Flag2 = VerifyCandidate(G', $e(u, \upsilon)$)
17: **end if**
18: **if** (Flag2 == True) **then**
19: $H_{k,i}$ = EnumComm(G', $e(u, \upsilon)$)
20: **end if**
21: i = i + 1
22: **output** $H_{k,i}$
23: **until** $((i = r) or (H = \emptyset))$



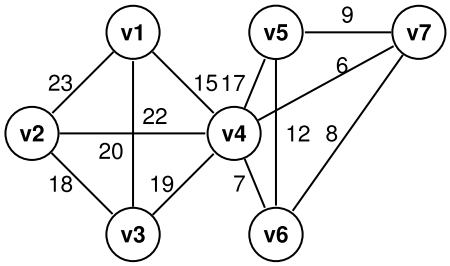
(a) The verified subgraph that contains the top-1 weighted 4-truss community.



(b) The top-1 Community with weight value=15.



(c) The verified subgraph that contains the top-2 4-truss community .



(d) Top-2 community with weight= 6.

FIGURE 3. **An example for discovering top-2 4-truss community.**

In the algorithm shown, all the sets of edges involved in a *k-truss* community for a given k (i.e., ∀ e ∈ E_k) are considered disconnected. Then edges with the highest weight are iteratively attached. The attached edge is moved from H to another graph instance G′. After that, UPDATESUPPORT PROCEDURE is called to update the support of the currently attached edge e and its affected set of edges e^* based on *lemma* 1. Using the conditions applied in UPDATESUPPORT PROCEDURE, we either conclude the existence or absence of a candidate community based on *lemma* 2. Nevertheless, concluding that we have a candidate community would require verification of its existence. VERIFYCANDIDATE PROCEDURE is called to verify the currently attached edge is part of a resulting community by checking the effect of all edges with support less than $k-2$ as detailed in subsection IV-A2. Finally, ENUMCOMM PROCEDURE is called to discover the verified community where the currently attached edge would be the community key-edge.

The $top-r\ weighted\ k-truss$ communities are correctly found by the proposed algorithm. For example, Given e is the least weight edge in the attached edges at a certain iteration. If the sup(e, G') is greater than or equal to $k-2$ and isn't affected by the previously attached edges with support less than $k-2$ (their removal wouldn't affect the recently attached edge support), then e is the key-edge of a discovered community $H_{k,i}$ where $H_{k,i}$ is a weighted k-truss community with rank i between the top-r communities. Edge e is the key-edge with the least weight between all edges in the community and represents the community weight (i.e., $w(e_{k,i}) < (w(e') : \forall e' \in E_{H_{k,i}})$)

Example 3: Consider the graph shown in FIGURE 2a, we need to find the top-2 weighted communities with k=4 (i.e., r=2, and k=4*). The graph in FIGURE 2a is first decomposed into a 4-truss community by removing the two edges* $\{(v_1, v_5), (v_3, v_6)\}$*. FIGURE 3a shows the subgraph that is found to be verified and contains the top-1 weighted 4-truss community after the consecutive edges attachment with the highest weight. FIGURE 3b shows the final resulting community. Similar to FIGURE 3a and FIGURE 3b, FIGURE 3c and FIGURE 3d show the verified subgraph and top-2 4-truss resulting community.*

1) UPDATE SUPPORT PROCEDURE UPON EDGE ATTACHMENT

Upon attachment of an edge $e(u,\upsilon)$ into a graph instance G′, the support of the attached edge is either updated to be equal

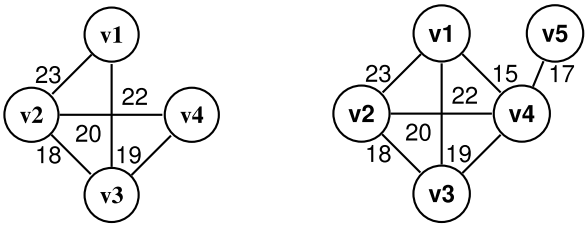
to the number of common neighbors between nodes u and υ ($sup(e, G') \neq 0$)) or the support of the attached edge is assigned zero if the ending points of the inserted edge $e(u, \upsilon)$ has no common neighbors (i.e., $nb(u) \cap nb(\upsilon) = \emptyset$) which means that no community will be detected upon this attachment. On the other hand, the support of the newly attached edge is changed when the ending points of the edge have common neighbors between them (i.e., $sup(e, G') = |nb(u) \cap nb(\upsilon)|$). According to lemma1, adjacent edges e^* support are changed by at most one. While updating the support of the affected edges e^*, the number of affected edges with support greater than or equal to $k-2$ is tracked. If the support of the newly attached edge e and at least $2*(k-2)$ from affected edges is $\geq k-2$, then a candidate community is found. Having at least $2*(k-2)$ affected edges with support $\geq k-2$ is a case that appears with the existence of a connected component according to lemma 2. Detecting such a case means that edge attachment has affected enough edges with a good number of connections to create a possible community that needs a verification step.

Algorithm 2 UPDATESUPPORT PROCEDURE

Input: An edge $e(u, \upsilon)$, G', and k
Output: Candidate Community State

```
Initialize min_sup=0
Initialize Flag=False
count=0
for each node w ∈ nb(u) ∩ nb(υ) do
    push w into a queue
end for
sup(e, G') = |nb(u) ∩ nb(υ)|
repeat
    pop a node w from queue;
    I.add(w)
    sup(e(w, u), G') = sup(e(w, u), G') + 1
    if (sup(e(w, u), G') ≥ k − 2) then
        count + +
    end if
    sup(e(w, υ), G') = sup(e(w, υ), G') + 1
    if (sup(e(w, υ), G') ≥ k − 2) then
        count + +
    end if
until (queue is empty)
if (sup(e, Gı) ≥ k − 2 and count ≥ 2 ∗ (k − 2)) then
    Flag = True
end if
return Flag
```

Example 4: Consider for example a graph in FIGURE 4a: upon attaching of the edge e(v2, v3), the support of the edge would be 2 which is equal to $k-2$ in our search for 4-truss community. On the other hand, the edge attachment has affected four edges where their new support is equal to one which is less than $k-2$. In other words, the number of affected edges with support $\geq k-2$ is zero which is less than $(2*(k-2))$. Then, the subgraph in FIGURE 4a is not a candidate solution. In FIGURE 4b, the subgraph in FIGURE 4a is enlarged by attaching the two edges $e(v_4, v_5)$ and $e(v_1, v_4)$. Upon attaching the edge $e(v_1, v_4)$, the subgraph in FIGURE 4b would be a candidate solution since the attached edge support is 2 which is equal to $k-2$. Besides, 4 affected edges which are equal to $(2*(k-2))$ edges have support value equal to 2 which is equal to $k-2$.



(a) A subgraph for a non-candidate solution (b) A subgraph for a candidate solution.

FIGURE 4. An example for non-candidate versus candidate subgraph for 4-truss community.

2) VERIFY AND ENUMERATE WEIGHTED K-TRUSS COMMUNITIES

Traditionally enumerating communities is tackled by decomposing the graph into *k-truss* communities which removes all edges that have minimum support less than $k-2$. Then Maximal Connected Component(MCC) procedure is called to discover connected components. Removing edges with minimum support less than $k-2$ doesn't represent any problem since the edges won't belong to the resulting communities in any case. On the other hand, the proposed algorithms discover communities by attaching edges incrementally. Removing edges to discover communities would ruin the sequence of the proposed algorithms. A new algorithm is devised to discover community containing the key-edge e without the need to do physical edges removal.

To overcome the physical edges removal problem, VERIFYCANDIDATE PROCEDURE is performed to check whether the currently attached edge and its set of affected edges have enough connections to output a community or not.

VERIFYCANDIDATE PROCEDURE tracks the state of the currently attached edge and its affected set of edges while logically removing edges with support less than k-2. If the currently attached edge isn't removed, then a community existence is verified with the currently attached edge denoted as the community key-edge. Once a community existence is verified, the connected component of this community has to be discovered. For example, the candidate solution in FIGURE 3a would have a verified community as removing edges with support less than k-2 (which is the edge $e(v_4, v_5)$) wouldn't affect the recently attached edge $e(v_1, v_4)$.

The next step is to discover the connected component existing in the verified subgraph. Two approaches for discovering connected components are proposed; Traditional Approach and Enumerate Weighted K-truss Communities Approach.

a: TRADITIONAL APPROACH FOR DISCOVERING CONNECTED COMPONENTS

The traditional algorithm traverses all the edges of the logically decomposed version of the graph starting from the community key-edge. During traversal, all edges with support $\geq k-2$ are added to the resulting community.

By observing the results of the traditional approach, we noticed that the resulting communities may be contained in each other. For example, FIGURE 3b shows the first resulting community which is contained in the resulting community shown in FIGURE 3d. The resulting community in FIGURE 3b has been already discovered and we don't have to go over the connections more than once to discover the largest community as in FIGURE 3d. This was the motivation to implement Enumerate weighted k-truss communities algorithm to incrementally discover communities.

b: ENUMERATE WEIGHTED K-TRUSS COMMUNITIES APPROACH

Enumerate weighted k-truss communities algorithm discovers a community through its key-edge. The key-edge and its affected set of edges will either form a complete resulting community or will be just a part of the resulting community. If the key-edge and its sequence of edges form a part of the community, then the remaining part consists of a subset of the previously discovered communities. In other words, the algorithm can detect the containment of one community in other communities. The relation between communities is discovered using the concept of the bitmap to define nodes states. The state of each node is stored in a bitmap where each bit would show whether the node already belongs to a discovered community or not. For a community H, the nodes of the key-edge and nodes of the affected edges resulting from `VERIFYCANDIDATE PROCEDURE` are used as the community initial nodes. Then, the algorithm retrieves the bitmaps of all community H initial nodes to do a logical ORing operation over all of them. The resulting bitmap shows which communities that the nodes in H belong to. Then, the nodes in H and the communities which nodes in H belong to -if there are any existing- are merged into H as a new top weighted community. The logical ORing operation allowed the algorithm to discover the containment of already discovered communities in the currently discovered community through one step using only the nodes bitmaps.

Algorithm 3 shows the pseudo-code for the enumerate weighted $k-truss$ communities algorithm. It takes as an input a key-edge e and a set of affected edges returned by `VERIFYCANDIDATE PROCEDURE`. In the algorithm presented, there are two global data structures to be shared among different runs; List_Keys which holds the key-edges incrementally (i.e. the size of List_keys indicates the number of communities observed till the current step), and dic_BitMaps which holds the affected nodes for each key-edge e and their bitmaps. The bitmap defined for each node in the affected nodes is of size r (number of communities).

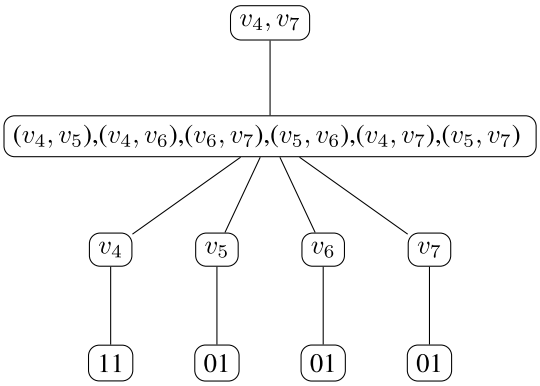
Algorithm 3 EnumComm

Input: *AffectedEdges*, and e
Output: A weighted Community H

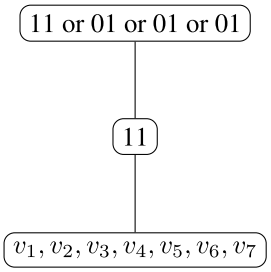
```
1: List_Key.add(e)
2: Affected_Nodes=Nodes(AffectedEdges)
3: Initialize H = Affected_Nodes
4: TempBitArray = bitarray()
5: for each node u ∈ Affected_Nodes do
6:     if (dic_BitMaps.has(u)) then
7:         TempBitArray = TempBitArray | dic_BitMaps[u]
8:         dic_BitMaps[u][index of e in List_Key]=1
9:     end if
10:    if (dic_BitMaps.has(u)= False) then
11:        Define BitArray_u=bitMap()
12:        BitArray_u[index of e in List_Key]=1
13:        dic_BitMaps.add[u]=BitArray_u
14:    end if
15: end for
16: i=0
17: repeat
18:    if (TempBitArray[i]=1) then
19:        H.append(output_Communities[List_Key(i)])
20:    end if
21:    i=i+1
22: until (i >= index of e in List_Key)
23: output_Communities.add(H)
```

In each bitmap of the affected nodes, the bit value corresponding to the index of the key-edge community is updated to 1. First, community H is initialized with the list of affected nodes returned by `VERIFYCANDIDATE PROCEDURE` in addition to the nodes of the key-edge. In other words, the list of affected nodes is initially added to be part of the output community H. From step3 to step14, for each node in the affected list, the existence in the dic_BitMaps is checked. If the bitmap of node u already exists, then its bitmap is retrieved for an incremental logical ORing. Bitmap existence means that the node has appeared at least once in the already discovered communities. Note that the incremental ORing between affected nodes' bitmaps ensures identifying all the already discovered communities which contain any of the affected nodes. If a node u does not exist in the dic_BitMaps, then a bitmap is defined with 1 in the bit index corresponding to the index of e in the List_Keys where e is the key-edge currently being discovered community index. The newly defined bitmap is added to dic_BitMaps. From step12 to step15, all the already discovered communities which contain any of the affected nodes are retrieved using the bits whose values are equal to 1 in the output bitmap of the logical ORing operation. Once a community is retrieved, its nodes are added to the currently discovered community.

Example 5: In this example enumerating the top-2 4-truss communities in FIGURE 3d is shown. The key-edge



(a) Simulation of keyedge removal and nodes bitmaps.



(b) The logical oring operation and output community

FIGURE 5. Enumerating community example.

of this community is $e(v_4, v_7)$ and the set of edges are {(v_4, v_5),(v_4, v_6),(v_6, v_7),(v_5, v_6),(v_5, v_7)} as shown in FIGURE 5a. The bitmap of each vertex of the affected edges is shown in FIGURE 5, the bitmap of v_4 has ones in the first and second bit which means that v_4 belongs to the first and second community. The remaining set of vertices' bitmaps have 1 only in the second bit. The vertex v_4 would be the bridge between the first and the currently discovered second community. FIGURE 5b shows the output of the ORing operations between nodes' bitmaps, the resulting community bitmap has two ones in the first and second bit. The first bit is referring to the first community already discovered and thus all vertices are added to the currently discovered second community. The top-2 community will contain the set of vertices {$v_1, v_2, v_3, v_4, v_5, v_6, v_7$}as shown in FIGURE 5b.

B. WEIGHT SENSITIVE LOCAL SEARCH

So far the algorithms proposed in the literature, either the recent one (OnlineAll) proposed in [11], or the newly one presented in section IV-A, discover the top-weighted communities using different ways but both initially start with the detection of the maximal *k-truss* community. While BACKWARD ALGORITHM performance is much better than OnlineAll algorithm as will be shown in section V, it still suffers from the long time that it takes in the decomposition step of the whole graph. In addition, the graph has to fit in the main memory to do the decomposition step. This may be a constraint if the size of the graph is too large to fit in the main memory and consequently BACKWARD ALGORITHM will not be able to discover top weighted communities.

In this section, the WEIGHT−SENSITIVE LOCAL SEARCH ALGORITHM (WSLSA) is proposed to discover the *top-r weighted k-truss* communities from graph G without the need to decompose the whole graph as an initial step. The idea behind WSLSA is to search for the top weighted communities using a small portion of the graph rather than the whole graph. And the decomposition step is done only on the part of the graph that contains the top weighted communities. Thus, the WSLSA can overcome the problem when the graph is too large to fit in main memory. WSLSA can be run while the graph needed is in main memory or disk resident. The main challenge of the WSLSA is determining which edges are used to start the search due to the availability of the whole graph rather than the decomposed version of the graph. WSLSA tackles this challenge by assuming that the edges of the graph are pre-sorted in decreasing order of their weights and starts the search from the edge with the highest weight. Then, the communities are built by incrementally adding edges in decreasing order with respect to their weight. Edges with highest weight will be added one after another and with each addition the support of the affected edges are updated. Once we reach a candidate for the k-truss community, a logical decomposition step is performed on a small portion of the graph that was built so far using the edges with the highest weight to verify the existence of k-truss community.

The algorithm initially defines an empty graph instance G' to hold the resulting communities. The resulting communities are enumerated while attaching the sorted edges with the highest weight following the same steps presented in the BACKWARD ALGORITHM algorithm. At any given edge attachment step t, if the edge attachment e_t increases the minimum support to be equal or greater than k-2 then G' contains one or more communities induced by the set of edges with weight at least $w(e_t)$. The motivation is to discover the same communities discovered by the BACKWARD ALGORITHM more efficiently.

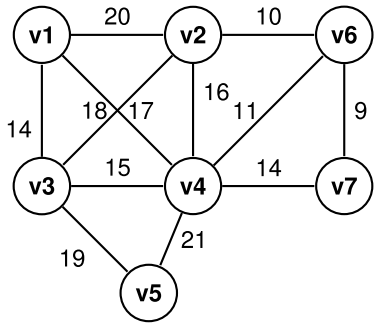
Algorithm 4 shows the pseudo-code of the (WSLSA). In the algorithm shown, an empty instance G' is initialized as an empty graph. Iteratively, edges are added from the sorted list to G'. Upon the existence of common neighbors between ending points of the newly added edge, the UPDATESUPPORT PROCEDURE is called to update the support of the newly added edge and the affected set of edges. Then, UPDATESUPPORT PROCEDURE checks the existence of a candidate solution using the support of the affected set of edges based on Lemma 2. Once a candidate solution is found, a verification of its existence is performed using VERIFYCANDIDATE PROCEDURE. Then, ENUMCOMM PROCEDURE is called to discover the nodes of the verified community. Given its similarity with the BACKWARD ALGORITHM, the WSLSA eventually discovers the correct *top-r weighted k-truss* communities. For example, given the graph in FIGURE 2a, the WSLSA discovers the

Algorithm 4 WEIGHT-SENSITIVE LOCAL SEARCH ALGORITHM (WSLSA)

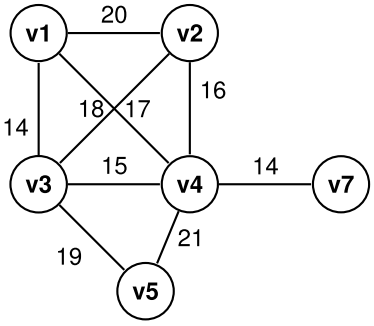
Input: An undirected graph G = (V, E, ω) and two parameters r, k

Output: The top-r weighted k-truss communities

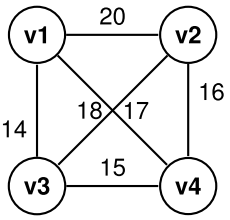
1: Initialize $Graph\_Instance(G') = \emptyset$
2: **repeat**
3: Find $e(u,\upsilon) \in E_H$ with the highest weight edge
4: add e to G'
5: **if** $(nb(u) \cap nb(\upsilon) \neq \emptyset)$ **then**
6: Flag1 = UpdateSupport(G', $e(u, \upsilon)$)
7: **end if**
8: **if** (Flag1 == True) **then**
9: Flag2 = VerifyCandidate(G', $e(u, \upsilon)$)
10: **end if**
11: **if** (Flag2 == True) **then**
12: $H_{k,i}$ = EnumComm(G', $e(u, \upsilon)$)
13: **end if**
14: i = i+1
15: **Output**$H_{k,i}$
16: **until** $((i = r) or (G' = G))$



(a) Graph Example.



(b) A subgraph that contains a candidate solution that is required to be verified.



(c) A verified 4-truss community with the highest weight.

FIGURE 6. A running example for WSLSA.

top-2 weighted community presented in FIGURE 3b and FIGURE 3d.

Example 6: To illustrate (WSLSA) *in a better way, consider the graph example in FIGURE 6a as we need to discover top weighted community with $k = 4$ (top weighted 4-truss community). In FIGURE 6b, edges with the highest weight are added one after another and the support of the edges are updated incrementally until the edge (v_1, v_3) is added. Once the edge (v_1, v_3) is added, a candidate solution is found since the subgraph in FIGURE 6b satisfies lemma2. Then* VERIFYCANDIDATE PROCEDURE *is performed to verify the existence of a community with key-edge (v_1, v_3). As* VERIFYCANDIDATE PROCEDURE *does logical decomposition on the subgraph in FIGURE 6b, all edges with support less than $k - 2$ are logically removed. Edges (v_3, v_5), (v_4, v_5), (v_4, v_7) will be logically removed since their trussness is less than 4. FIGURE 6c shows the top-weighted 4-truss community with weight=14.*

TABLE 1. Datasets description.

Graph	|V|	|E|	k_{max}
Wiki-Vote	8K	200K	25
Email	37K	200K	20
DBLP	317K	1M	100
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

V. EXPERIMENTS

Extensive experiments are conducted to evaluate the proposed algorithms in terms of their efficiency against the state-of-the-art Basic algorithm proposed in [11]. The Basic algorithm as described in related work section, is one of the most recent algorithms to perform online weighted k-truss community search. Experiments are conducted on five publicly available datasets. All algorithms are implemented using Python environment. In addition, all experiments are conducted on a machine with an Intel i5 2.5GHz CPU and 8GB main memory.

A. DATASETS

Eight public datasets as availed in [35] are used in our experiments to evaluate the proposed algorithms. Graphs of each dataset described in TABLE 1, where V represents the number of vertices, E represents the number of edges, and k_{max} represents the maximum k-truss that can be found in the graph. For all graphs, the weight of each edge $e(u, \upsilon)$ is represented by the number of common neighbors between nodes u, and υ. Either the edge weight is calculated or given in the graph, the proposed algorithms are expected to extract the top-weighted communities. As clarified in TABLE 1, the datasets used for evaluation vary from large to small size. The largest datasets are the Wiki-Talk, Skitter, LiveJournal and Orkut graphs. Wiki-Talk, LiveJournal, and Orkut are social networks, whereas Wiki-Talk represents communication network between Wikipedia users, LiveJournal and Orkut represent frienship networks between users and Skitter represents internet topology network. The moderate size dataset are DBLP, and Youtube, where the DBLP graph represents the collaboration network between authors, and Youtube is a social network that represents the friendship

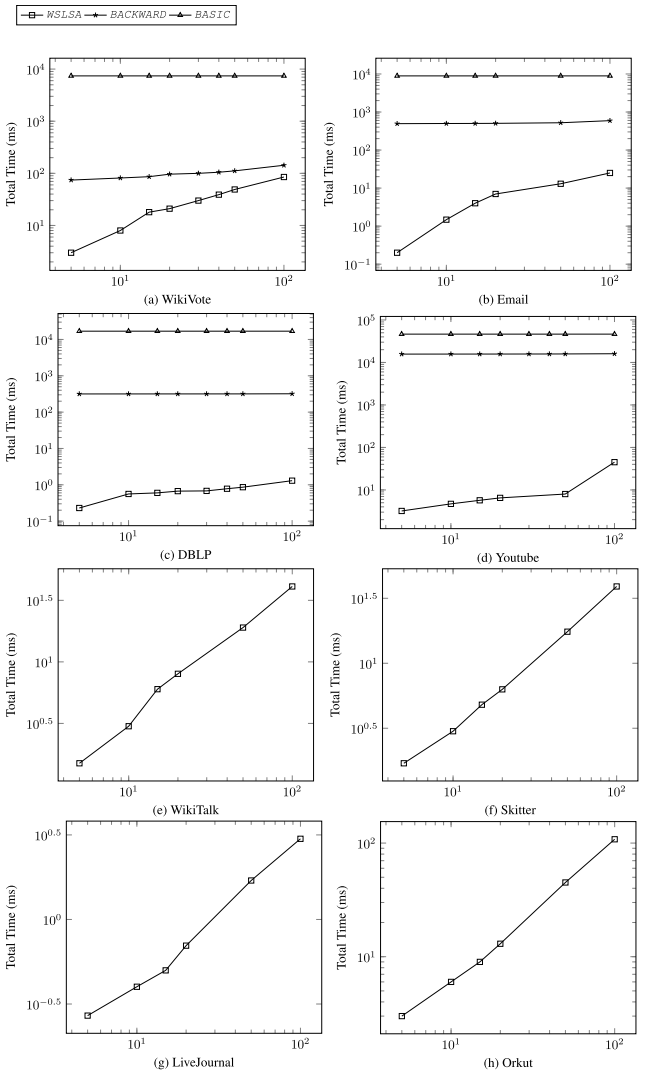


FIGURE 7. k=10 vary r.

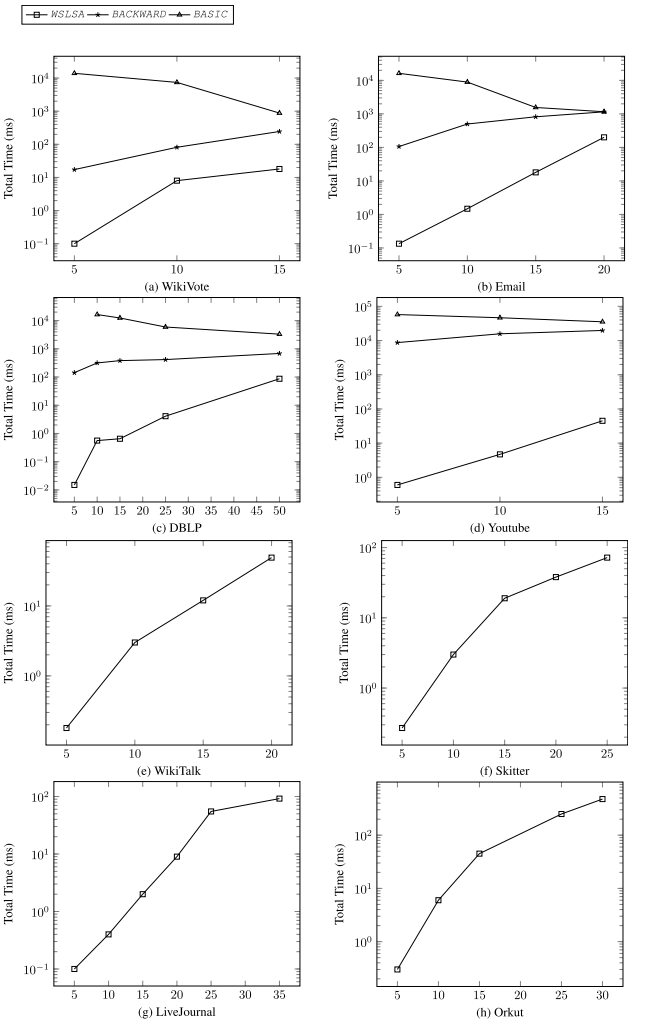


FIGURE 8. r=10 vary k.

between Youtube users. The last two datasets Wiki-Vote and Email-Enron are of small sizes, where Wiki-Vote is the voting network between Wikipedia users and Email-Enron is the email communication network from Enron.

B. QUERY PARAMETERS

There are two query parameters k and r, k is chosen from {3,5,10,15,20,50}, and r is chosen from {5,10,15,25,50,100} where r is the number of top communities to be retrieved, and k is cohesiveness measure of the community.

C. EXPERIMENTAL RESULTS

In the evaluation, the proposed algorithms are evaluated against BASIC algorithm by varying k while using default r with value 10 and varying r while k with a default value 10. The processing time of the algorithms by varying r and k=10 is shown in FIGURE 7 while the processing time of the algorithms by varying k and r=10 is shown in FIGURE 8.

Generally FIGURE 7 and FIGURE 8 show that the WSLSA runs significantly faster than BACKWARD and BASIC algorithms and that's because WSLSA visits a small portion of the graph whereas the BACKWARD and BASIC algorithms decompose the entire graph as an initial step. In addition, BACKWARD ALGORITHM performance is still much better than the BASIC algorithm as it discovers communities incrementally. It is also noted that, WSLSA takes much smaller processing time for small values of k and r regardless of the graph size. This is related to how WSLSA finds communities through adding edges with the highest weight incrementally. With small values of k and r, it is guaranteed that the required communities are discovered quickly even for very large graphs.

As illustrated in FIGURE 7 by increasing r WSLSA takes more time as it processes larger subgraph to find more communities however it still runs significantly faster than the other two algorithms. On the other hand, BACKWARD and BASIC algorithms take almost constant time to discover different number of communities. The reason behind that, in the BACKWARD algorithm all edges that don't belong to any

community with required k are removed while decomposing the entire graph first. Once the first community is discovered, all subsequent communities are usually contained in each other. Thus, discovering communities requires almost constant time as most of them are retrieved from bitmaps rather than finding a connected component for each community.

From FIGURE 8, it's noted that WSLSA takes longer processing time for k when it approaches max k of the graph and large r but still outperforms BACKWARD and BASIC algorithms. This is related to the large number of candidates generated during discovering communities from graphs with large r and almost max k where each candidate is required to be verified. Both generation and verification of a large number of candidates increase the processing time.

It is also clear that the difference between BACKWARD and WSLS algorithms in terms of processing time is significant. However, the processing time for discovering *top-r* communities in BACKWARD ALGORITHM without decomposition time is faster than WSLSA. The reason behind this observation is that the BACKWARD ALGORITHM has to prune all edges with trussness less than k as a first step while WSLSA has to process those edges before discovering each community. Generally, BACKWARD ALGORITHM is the best solution and leads to faster results if decomposition step cost is low or the decomposed version of the graph is already available and stored.

Concerning the Wiki-Talk, LiveJournal, Skitter, and Orkut dataset, only WSLSA was run on each graph and processing time was obtained which included reading time for the graph edges from the disk. On the other hand, BACKWARD and BASIC algorithms were not run on those datasets given the large size of the graphs and the inability to decompose such large graphs in the main memory. The reported results for such large graphs prove the efficiency of the WSLSA. For instance, WSLSA required 32 seconds to get top-300 weighted 5-truss communities from Orkut dataset while Basic algorithm required 10000 seconds to get the top-40 weighted 5-truss communities as reported in [11]. In addition, the results show the advantage for WSLSA over BACKWARD and BASIC algorithms as it was able to discover communities from such large graphs without the need for extreme computational power. Furthermore, the incremental processing behavior of WSLSA can be utilized to process much larger disk-based graphs rather than memory-based ones.

VI. CONCLUSION AND FUTURE WORK

In this paper, weight-based k-truss community search problem was investigated. WSLSA and BACKWARD ALGORITHM were proposed to discover *top-r weighted k-truss* communities from graphs. Both algorithms utilize the concepts of local search and bitmaps to discover communities. The algorithms were evaluated using different graphs with different sizes against the state-of-the-art Basic algorithm. Experimental results showed that WSLSA outperforms both the algorithms BACKWARD and Basic significantly. As future work, it's possible to extend the algorithms to include the vertices attributes as an extra dimension to discover homogeneous top-weighted communities. Also, expanding the algorithms to discover top-weighted communities where each edge is weighted with multiple weights is another interesting direction to follow. Another future direction is to optimize the proposed algorithms to reduce the processing time especially when the required trussness approaches the graph maximum trussness.

REFERENCES

[1] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 69, no. 2, Feb. 2004, Art. no. 026113.

[2] J. Cohen, "Trusses: Cohesive subgraphs for social network analysis," Nat. Secur. Agency, Fort Meade, MD, USA, Tech. Rep., 2008, pp. 1–3, vol. 16.

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

[4] R.-H. Li, L. Qin, J. X. Yu, and R. Mao, "Influential community search in large networks," *Proc. VLDB Endowment*, vol. 8, no. 5, pp. 509–520, Jan. 2015.

[5] W. Cui, Y. Xiao, H. Wang, and W. Wang, "Local search of communities in large graphs," in *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, New York, NY, USA, 2014, pp. 991–1002. [Online]. Available: http://doi.acm.org/10.1145/2588555.2612179

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

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

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

[9] S. Chen, R. Wei, D. Popova, and A. Thomo, "Efficient computation of importance based communities in Web-scale networks using a single machine," in *Proc. 25th ACM Int. Conf. Inf. Knowl. Manage.*, Oct. 2016, pp. 1553–1562.

[10] S. Fortunato, "Community detection in graphs," *Phys. Rep.*, vol. 486, nos. 3–5, pp. 75–174, Feb. 2010.

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

[12] A. Garas, P. Argyrakis, C. Rozenblat, M. Tomassini, and S. Havlin, "Worldwide spreading of economic crisis," *New J. Phys.*, vol. 12, no. 11, Nov. 2010, Art. no. 113043.

[13] L. Chang, W. Li, L. Qin, W. Zhang, and S. Yang, "pSCAN: Fast and exact structural graph clustering," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 2, pp. 387–401, Feb. 2017.

[14] J. Shao, Z. Han, Q. Yang, and T. Zhou, "Community detection based on distance dynamics," in *Proc. 21th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, New York, NY, USA, 2015, pp. 1075–1084. [Online]. Available: http://doi.acm.org/10.1145/2783258.2783301

[15] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," *Knowl. Inf. Syst.*, vol. 42, no. 1, pp. 181–213, Jan. 2015.

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

[17] P. R. Suaris and G. Kedem, "An algorithm for quadrisection and its application to standard cell placement," *IEEE Trans. Circuits Syst.*, vol. CAS-35, no. 3, pp. 294–303, Mar. 1988.

[18] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, vol. 49, no. 2, pp. 291–307, Feb. 1970.

[19] J. Cheng, Y. Ke, A. W.-C. Fu, J. X. Yu, and L. Zhu, "Finding maximal cliques in massive networks," *ACM Trans. Database Syst.*, vol. 36, no. 4, p. 21, 2011.

[20] J. Cheng, L. Zhu, Y. Ke, and S. Chu, "Fast algorithms for maximal clique enumeration with limited memory," in *Proc. 18th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, 2012, pp. 1240–1248.

[21] C. Tsourakakis, F. Bonchi, A. Gionis, F. Gullo, and M. Tsiarli, "Denser than the densest subgraph: Extracting optimal quasi-cliques with quality guarantees," in *Proc. 19th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, 2013, pp. 104–112.
[22] J. Cheng, Y. Ke, S. Chu, and M. T. Özsu, "Efficient core decomposition in massive networks," in *Proc. IEEE 27th Int. Conf. Data Eng.*, Apr. 2011, pp. 51–62.
[23] W. Khaouid, M. Barsky, V. Srinivasan, and A. Thomo, "K-core decomposition of large networks on a single PC," *Proc. VLDB Endowment*, vol. 9, no. 1, pp. 13–23, Sep. 2015.
[24] M. Charikar, "Greedy approximation algorithms for finding dense components in a graph," in *Proc. Int. Workshop Approximation Algorithms Combinat. Optim.* Berlin, Germany: Springer, 2000, pp. 84–95.
[25] *Finding a Maximum Density Subgraph*, Univ. California, Berkely, Berkeley, CA, USA, May 1984.
[26] L. Chang, J. X. Yu, L. Qin, X. Lin, C. Liu, and W. Liang, "Efficiently computing k-edge connected components via graph decomposition," in *Proc. Int. Conf. Manage. Data (SIGMOD)*, 2013, pp. 205–216.
[27] R. Zhou, C. Liu, J. X. Yu, W. Liang, B. Chen, and J. Li, "Finding maximal k-edge-connected subgraphs from a large graph," in *Proc. 15th Int. Conf. Extending Database Technol. (EDBT)*, 2012, pp. 480–491.
[28] M. Alemi and H. Haghighi, "KTMiner: Distributed k-truss detection in big graphs," *Inf. Syst.*, vol. 83, pp. 195–216, Jul. 2019.
[29] M. Sozio and A. Gionis, "The community-search problem and how to plan a successful cocktail party," in *Proc. 16th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, 2010, pp. 939–948.
[30] W. Cui, Y. Xiao, H. Wang, Y. Lu, and W. Wang, "Online search of overlapping communities," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2013, pp. 277–288.
[31] Y. Wu, R. Jin, J. Li, and X. Zhang, "Robust local community detection: On free rider effect and its elimination," *Proc. VLDB Endowment*, vol. 8, no. 7, pp. 798–809, Feb. 2015.
[32] F. Bi, L. Chang, X. Lin, and W. Zhang, "An optimal and progressive approach to online search of top-k influential communities," *Proc. VLDB Endowment*, vol. 11, no. 9, pp. 1056–1068, May 2018.
[33] A. E. Sariyüce and A. Pinar, "Fast hierarchy construction for dense subgraphs," *Proc. VLDB Endowment*, vol. 10, no. 3, pp. 97–108, Nov. 2016, doi: 10.14778/3021924.3021927.
[34] A. E. Sarıyüce, B. Gedik, G. Jacques-Silva, K.-L. Wu, and U. V. Çatalyürek, "Incremental k-core decomposition: Algorithms and evaluation," *Int. J. Very Large Data Bases*, vol. 25, no. 3, pp. 425–447, Jun. 2016.
[35] *SNAP Networks Datasets Website*. Accessed: Jun. 2019. [Online]. Available: https://snap.stanford.edu/data/

WAFAA M. A. HABIB received the B.Sc. and M.Sc. degrees from the Department of Information Systems, Faculty of Computers and Artificial Intelligence, Cairo University, in 2010 and 2014, respectively, where she is currently pursuing the Ph.D. degree in information systems with the Faculty of Computers and Artificial Intelligence.

HODA M. O. MOKHTAR received the B.Sc. (Hons.) and M.Sc. degrees from the Department of Computer Engineering, Faculty of Engineering, Cairo University, in 1997 and 2000, respectively, and the Ph.D. degree in computer science from the University of California Santa Barbara, in 2005. She has taught multiple courses for the undergraduate and graduate levels and supervised a number of master's and Ph.D. theses with the Faculty of Computers and Artificial Intelligence, Cairo University, where she is currently the Chair of the Information Systems Department. She has participated in several national committees. Her research interests include big data analytics, data warehousing, data mining, database systems, social network analysis, bioinformatics, and web services. She received the Scholarship and the Dean's Fellowship from the Computer Science Department, University of California Santa Barbara (UCSB), in 2000. She also received the multiple awards and certificates for her academic achievements.

MOHAMED E. EL-SHARKAWI received the B.Sc. degree in systems and computer engineering from the Faculty of Engineering, Al-Azhar University, Cairo, Egypt, and the M.Eng. and Ph.D. degrees in computer science and communication engineering from the Faculty of Engineering, Kyushu University, Japan. He is currently a Professor with the Department of Information Systems, Faculty of Computers and Artificial Intelligence, Cairo University. His research interests include data engineering, including query processing, temporal databases, and social network analysis.