Approximate Closest Community Search in Networks

Xin Huang[†], Laks V.S. Lakshmanan[†], Jeffrey Xu Yu[‡], Hong Cheng[‡]

[†] *University of British Columbia,* [‡]*The Chinese University of Hong Kong*
{xin0,laks}@cs.ubc.ca, {yu, hcheng}@se.cuhk.edu.hk

ABSTRACT

Recently, there has been significant interest in the study of the community search problem in social and information networks: given one or more query nodes, find densely connected communities containing the query nodes. However, most existing studies do not address the "free rider" issue, that is, nodes far away from query nodes and irrelevant to them are included in the detected community. Some state-of-the-art models have attempted to address this issue, but not only are their formulated problems NP-hard, they do not admit any approximations without restrictive assumptions, which may not always hold in practice.

In this paper, given an undirected graph G and a set of query nodes Q, we study community search using the k-truss based community model. We formulate our problem of finding a *closest truss community* (CTC), as finding a connected k-truss subgraph with the largest k that contains Q, and has the minimum diameter among such subgraphs. We prove this problem is NP-hard. Furthermore, it is NP-hard to approximate the problem within a factor $(2-\varepsilon)$, for any $\varepsilon > 0$. However, we develop a greedy algorithmic framework, which first finds a CTC containing Q, and then iteratively removes the furthest nodes from Q, from the graph. The method achieves 2-approximation to the optimal solution. To further improve the efficiency, we make use of a compact truss index and develop efficient algorithms for k-truss identification and maintenance as nodes get eliminated. In addition, using bulk deletion optimization and local exploration strategies, we propose two more efficient algorithms. One of them trades some approximation quality for efficiency while the other is a very efficient heuristic. Extensive experiments on 6 real-world networks show the effectiveness and efficiency of our community model and search algorithms.

1. INTRODUCTION

Community structures naturally exist in many real-world networks such as social, biological, collaboration, and communication networks. The task of community detection is to identify all communities in a network, which is a fundamental and well-studied problem in the literature. Recently, several papers have studied a related but different problem called *community search*, which is to
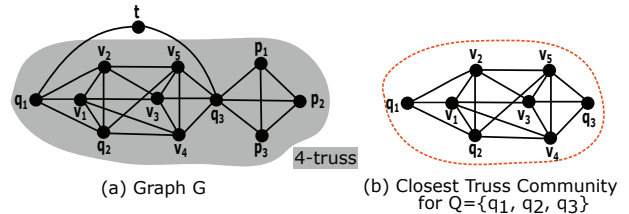
Figure 1: Closest truss community example

find the community containing a given set of query nodes. The need for community search naturally arises in many real application scenarios, where one is motivated by the discovery of the communities in which given query nodes participate. Since the communities defined by different nodes in a network may be quite different, community search with query nodes opens up the prospects of user-centered and personalized search, with the potential of the answers being more meaningful to a user [16]. As just one example, in a social network, the community formed by a person's high school classmates can be significantly different from the community formed by her family members which in turn can be quite different from the one formed by her colleagues [20].

Various community models have been proposed based on different dense subgraph structures such as k-core [25, 9, 19] , k-truss [16], quasi-clique [8], weighted densest subgraph [28], to name a few major examples. Of these, the k-truss as a definition of cohesive subgraph of a graph G, requires that each edge be contained in at least $(k - 2)$ triangles within this subgraph. Consider the graph G in Figure 1; in the subgraph in the whole grey region (i.e., excluding the node t), each edge is contained in two triangles. Thus, the subgraph is a 4-truss. It is well known that most of real-world social networks are triangle-based, which always have high local clustering coefficient. Triangles are known as the fundamental building blocks of networks [27]. In a social network, a triangle indicates two friends have a common friend, which shows a strong and stable relationship among three friends. Intuitively, the more common friends two people have, the stronger their relationship. In a k-truss, each pair of friends is "endorsed" by at least $(k-2)$ common friends. Thus, a k-truss with a large value of k signifies strong inner-connections between members of the subgraph. Huang et al. [16] proposed a community model based on the notion of k-truss as follows. Given *one* query node q and a parameter k, a k-truss community containing q is a maximal k-truss containing q, in which each edge is "triangle connected" with other edges. Triangle connectivity is strictly stronger than connectivity. The k-truss community model works well to find all overlapping communities containing a query node q. It is natural to search for communities containing a *set* of query nodes in real applications, and the above community model, extended for multiple query nodes, has

the following limitations. Due to the strict requirement of triangle connectivity constraint, the model may fail to discover any community for query nodes. For example, for the graph of Figure 1(a), and query nodes $Q = \{v_4, q_3, p_1\}$ the above k-truss community model cannot find a qualified community for any k, since the edges (v_4, q_3) and (q_3, p_1) are not triangle connected in any k-truss. A detailed comparison of various community search models and techniques can be found in the Section 7.

In this paper, we study the problem of *close community search*, i.e., given a set of query nodes, find a dense connected subgraph that contains the query nodes, in which nodes are close to each other. As a qualifying cohesive structure, we use the notion of k-truss for modeling a densely connected community, which inherits several good structural properties, such as k-edge connectivity, bounded diameter and hierarchical structure. In addition, to ensure every node included in the community is tightly related to query nodes and other nodes included in the community reported, we use graph diameter to measure the closeness of all nodes in the community. Thus, based on k-truss and graph diameter, we propose a novel community model as **closest truss community (CTC)**, which requires that the all query nodes are connected in this community, the graph structure is a k-truss with the largest trussness k. In general, several such candidate communities may exist. Some of them may suffer from the so-called "free rider effect" formally defined and studied in [28]. While we discuss this in detail in Section 3.2, we illustrate it with an example here. In Figure 1(a), for the query nodes $\{q_1, q_2, q_3\}$, the subgraph shaded grey is a 4-truss containing the query nodes. It includes the nodes p_1, p_2, p_3 which are intuitively not relevant to the query nodes. Specifically, they are all far away from q_1 and can be regarded as "free riders". This 4-truss is said to suffer from the free rider effect. On the other hand, the subgraph without the nodes $\{p_1, p_2, p_3\}$ is also a 4-truss, it has the smallest diameter among all 4-trusses containing the query nodes, and does not suffer from the free rider effect. Motivated by this, we define a closest truss community as a connected k-truss with the largest k containing the query nodes and having the smallest diameter. We show that such a definition avoids the free rider effect. A connected k-truss with the largest k containing given query nodes can be found in polynomial time. However, as we show, finding such a k-truss with the minimum diameter is NP-hard and it is hard to approximate within a factor better than 2. Here, the approximation is w.r.t. the minimum diameter. On the other hand, we develop a greedy strategy for finding a CTC that delivers a 2-approximation to the optimal solution, thus essentially matching the lower bound. In order to make our algorithm scalable to large real networks, we propose two techniques. One of them is based on bulk deletion of nodes far away from query nodes. The second is a heuristic exploration of the local neighborhood of a Steiner tree containing the query nodes. The challenge here is that a naive application of Steiner trees may yield a k-truss with a low value of k, which is undesirable. We address this challenge by developing a new notion of distances based on edge trussness. Specifically, we make the following contributions in this paper.

- We propose a novel community search model called *closest truss community* (CTC) and motivate the problem of finding CTC containing given query nodes (Section 2).

- We analyze the structural and computational properties of CTC and show that it avoids the free rider effect, is NP-hard to compute exactly or to approximate within a factor of $(2 - \varepsilon)$, for any $\varepsilon > 0$ (Section 3).

- We develop a greedy 2-approximation algorithm for finding a CTC given set of query nodes. The algorithm is based on

Table 1: Frequently Used Notations

Notation	Description
$G = (V(G), E(G))$	An undirected and connected simple graph G
$n; m$	The number of vertices/edges in G
$N(v)$	The set of neighbors of v
$\sup_H(e)$	The *support* of edge e in H
$\tau(H)$	Trussness of graph H
$\tau(e)$	Trussness of edge e
$\tau(v)$	Trussness of vertex v
$\bar{\tau}(S)$	The maximum trussness of connected graphs containing S
$\text{diam}(H)$	The diameter of graph H
$\text{dist}_H(v, u)$	The shortest distance between v and u in H
$\text{dist}_H(R, Q)$	$\text{dist}_H(R, Q) = \max_{v \in R, u \in Q} \text{dist}_H(v, u)$

finding, in linear time, a connected k-truss with maximum k containing the query nodes, using a simple truss index. Then successively nodes far away from the query nodes are eliminated (Section 4).

- We further speed up CTC search in two ways: (1) we make use of a clever bulk deletion strategy and (2) find a Steiner tree of the query nodes and expand it into a k-truss by exploring the local neighborhood of the Steiner tree. The first of these slightly degrades the approximation factor while the second is a heuristic (Section 5).

- We extensively experiment with the various algorithms on 6 real networks. Our results show that our closest truss community model can efficiently and effectively discover the queried communities on real-world networks with ground-truth communities. (Section 6).

In Section 7, we present a detailed comparison with related work. In Section 7.1, we discuss alternative candidates for community models and provide a rationale for our design decisions. We summarize the paper in Section 8.

2. PROBLEM DEFINITION

We consider an undirected, unweighted simple graph $G = (V(G), E(G))$ with $n = |V(G)|$ vertices and $m = |E(G)|$ edges. We denote the set of neighbors of a vertex v by $N(v)$, i.e., $N(v) = \{u \in V : (v, u) \in E\}$, and the degree of v by $d(v) = |N(v)|$. We use $d_{max} = \max_{v \in V} d(v)$ to denote the maximum vertex degree in G. W.l.o.g we assume in this paper that the graph G we consider is connected. Note that this implies that $m \geq n - 1$. Table 1 summarizes the frequently used notations in the paper.

A *triangle* in G is a cycle of length 3. Let $u, v, w \in V$ be the three vertices on the cycle, then we denote this triangle by $\triangle_{uvw}$. The *support* of an edge $e(u, v) \in E$ in G, denoted $sup_G(e)$, is defined as $|\{\triangle_{uvw} : w \in V\}|$. When the context is obvious, we drop the subscript and denote the support as $sup(e)$. Based on the definition of k-truss [6, 27], we define a connected k-truss as follows.

DEFINITION 1 (CONNECTED K-TRUSS). *Given a graph G and an integer k, a connected k-truss is a connected subgraph $H \subseteq G$, such that $\forall e \in E(H), sup_H(e) \geq (k - 2)$.*

Intuitively, a connected k-truss is a connected subgraph such that each edge (u, v) in the subgraph is "endorsed" by $k - 2$ common neighbors of u and v [6]. In a connected k-truss graph, each node has degree at least $k - 1$ and a connected k-truss is also a $(k - 1)$-core [2]. Next, we define the *trussness* of a subgraph, an edge, and a vertex as follows.

DEFINITION 2 (TRUSSNESS). *The trussness of a subgraph $H \subseteq G$ is the minimum support of an edge in H plus 2, i.e., $\tau(H) =$*

$2 + \min_{e \in E(H)} \{sup_H(e)\}$. *The trussness of an edge $e \in E(G)$ is $\tau(e) = \max_{H \subseteq G \wedge e \in E(H)} \{\tau(H)\}$. The trussness of a vertex $v \in V(G)$ is $\tau(v) = \max_{H \subseteq G \wedge v \in V(H)} \{\tau(H)\}$.*

Consider the graph G in Figure 1(a). Edge $e(q_2, v_2)$ is contained in three triangles $\triangle_{q_2 v_2 q_1}$, $\triangle_{q_2 v_2 v_1}$ and $\triangle_{q_2 v_2 v_5}$, thus its support is $sup_G(e(q_2, v_2)) = 3$. Suppose H is the triangle $\triangle_{q_2 v_2 q_1}$, then the trussness of the subgraph H is $\tau(H) = 2 + \min_{e \in H} sup_H(e) = 3$, since each edge is contained in one triangle in H. The trussness of the edge $e(q_2, v_2)$ is 4, because in the induced subgraph on vertices $\{q_1, q_2, v_1, v_2\}$, each edge is contained in two triangles in the subgraph and any subgraph H containing $e(q_2, v_2)$ has $\tau(H) \leq 4$, i.e., $\tau(e(q_2, v_2)) = \max_{H \subseteq G \wedge e \in E(H)} \{\tau(H)\} = 4$. Note that the trussness of an edge e of a graph G could be less than $sup_G(e) + 2$, e.g., $\tau(e(q_2, v_2)) = 4 < 5 = sup(e(q_2, v_2)) + 2$. Moreover, the vertex trussness of q_2 is also 4, i.e. $\tau(q_2) = 4$.

For a set of vertices $S \subseteq V(G)$, we use $\bar{\tau}(S)$ to denote the maximum trussness of a *connected* subgraph H containing S, i.e., $\bar{\tau}(S) = \max_{S \subseteq H \subseteq G \wedge H \text{is connected}} \{\tau(H)\}$. Notice that by definition, for $S = \emptyset$, $\bar{\tau}(\emptyset)$ is the maximum trussness of any edge in G. In Figure 1(a), the whole subgraph in the grey region is a 4-truss. There exists no 5-truss in G, and $\bar{\tau}(\emptyset) = 4$. We will make use of $\bar{\tau}(\emptyset)$ in Section 5.

For two nodes $u, v \in G$, we denote by $\text{dist}_G(u, v)$ the length of the shortest path between u and v in G, where $\text{dist}_G(u, v) = +\infty$ if u and v are not connected. We make use of the notions of graph query distance and diameter in the rest of the paper.

DEFINITION 3 (QUERY DISTANCE). *Given a graph G and a set of query nodes $Q \subset V$, for each vertex $v \in G$, the vertex query distance of v is the maximum length of a shortest path from v to a query node $q \in Q$, i.e., $\text{dist}_G(v, Q) = \max_{q \in Q} \text{dist}_G(v, q)$. For a subgraph $H \subseteq G$, the graph query distance of H is defined as $\text{dist}_G(H, Q) = \max_{u \in H} \text{dist}_G(u, Q) = \max_{u \in H, q \in Q} \text{dist}_G(u, q)$.*

DEFINITION 4 (GRAPH DIAMETER). *The diameter of a graph G is defined as the maximum length of a shortest path in G, i.e., $\text{diam}(G) = \max_{u, v \in G} \{\text{dist}_G(u, v)\}$.*

For the graph G in Figure 1(a) and $Q = \{q_2, q_3\}$, the vertex query distance of v_2 is $\text{dist}_G(v_2, Q) = \max_{q \in Q} \{dist_G(v_2, q)\} = 2$, since $\text{dist}_G(v_2, q_3) = 2$ and $\text{dist}_G(v_2, q_2) = 1$. Let H be the subgraph of Figure 1(a) shaded in grey. Then query distance of H is $\text{dist}_G(H, Q) = 3$. The diameter of H is $\text{diam}(H) = 4$.

On the basis of the definitions of k-truss and graph diameter, we define the *closest truss community* in a graph G as follows.

DEFINITION 5 (CLOSEST TRUSS COMMUNITY). *Given a graph G and a set of query nodes Q, G' is a closest truss community (CTC), if G' satisfies the following two conditions:*

(1) Connected k-Truss. *G' is a connected k-truss containing Q with the largest k, i.e., $Q \subseteq G' \subseteq G$ and $\forall e \in E(G')$, $sup(e) \geq k - 2$;*

(2) Smallest Diameter. *G' is a subgraph of smallest diameter satisfying condition (1). That is, $\nexists G'' \subseteq G$, such that $diam(G'') < diam(G')$, and G'' satisfies condition (1).*

Condition (1) requires that the closest community containing the query nodes Q be densely connected. In addition, Condition (2) makes sure that each node is as close as possible to every other node in the community, including the query nodes. We next illustrate the notion of CTC as well as the consequence of considering Conditions (1) and (2) in different order.

EXAMPLE 1. In Definition 5, we firstly consider the connected k-truss of G containing query nodes with the largest trussness, and then among such subgraphs, regard the one with the smallest diameter as the closest truss community. Consider the graph G in Figure 1(a), and $Q = \{q_1, q_2, q_3\}$; the subgraph in the region shaded grey is a 4-truss containing Q, and is a subgraph with the largest trussness that contains Q, and has diameter 4. Notice that in Figure 1(a), although the nodes p_1, p_2, p_3 belong to the 4-truss and are strongly connected with q_3, they are far away from the query node q_1. Figure 1(b) shows another 4-truss containing Q but not p_1, p_2, p_3, and its diameter is 3. It can be verified that this is the 4-truss with the smallest diameter. Thus, by Condition (2) of Definition 5, the 4-truss graph in Figure 1(a) will not be regarded the closest truss community, whereas the one in Figure 1(b) is indeed the CTC. Intuitively, the nodes p_1, p_2, p_3 are "free riders" that belong to a community defined only using Condition (1), and are avoided by Condition (2). We will see in Section 3.2 that the definition of CTC above avoids the so-called "free rider effect". □

EXAMPLE 2. Suppose we apply the conditions in Definition 5 in the opposite order. That is, we first minimize the diameter among connected subgraphs of G containing Q and look for the k-truss subgraph with the largest k among those. Firstly, we find that the cycle of $\{(q_1, t), (t, q_3), (q_3, v_4), (v_4, q_2), (q_2, q_1)\}$ is the connected subgraph containing Q with the smallest diameter 2. Then, we find that this cycle is also the k-truss subgraph with the largest k containing itself. However, it is only a 2-truss, which has a loosely connected structure compared to Figure 1(b). This justifies the choice of the order in which Conditions (1) and (2) should be applied. □

We discuss several natural candidates for community models in Section 7.1 and provide a rationale for our design decisions. We have a choice between minimizing diameter or minimizing query distance. We address this choice in Section 3.2: Example 3 illustrates the value added by minimizing the diameter over minimizing just the query distance. The problem of **closest truss community (CTC) search** studied in this paper is stated as follows.

PROBLEM 1 (CTC-Problem). *Given a graph $G(V, E)$ and a set of query vertices $Q = \{v_1, ..., v_r\} \subseteq V$, find a closest truss community containing Q.*

3. PROBLEM ANALYSIS

3.1 Structural Properties

Since our closest truss community model is based on the concept of k-truss, the communities capture good structral properties of k-truss, such as *k-edge-connected* and *hierarchical structure*. In addition, since CTC is required to have minimum diameter, it also has *bounded diameter*. As a result, CTC avoids the "free rider effect" [25, 28] (see Section 3.2).

Small diameter, k-edge-connected, hierarchical structure. First, the diameter of a connected k-truss with n vertices is no more than $\lfloor \frac{2n-2}{k} \rfloor$ [6]. The diameter of a community is considered as an important feature of a community [11]. Moreover, a k-truss community is $(k-1)$-edge-connected [6], as it remains connected whenever fewer than $k-1$ edges are removed [13]. In addition, k-truss based community has *hierarchical structure* that represents the cores of a community at different levels of granularity [16], that is, k-truss is always contained in the $(k-1)$-truss for any $k \geq 3$.

Largest k. We have a trivial upper bound on the maximum possible trussness of a connected k-truss containing the query nodes.

LEMMA 1. *For a connected k-truss H satisfying definition of CTC for Q, we have $k \leq \min\{\tau(q_1), ..., \tau(q_r)\}$ holds.*

PROOF. First, we have $Q \subseteq H$. For each node $q \in Q$, q cannot be contained in a k-truss in G, whenever $k > \tau(q)$. Thus, the fact that H is a k-truss subgraph containing Q implies that $k \leq \min\{\tau(q_1), ..., \tau(q_r)\}$. $\square$

Lower and upper bounds on diameter. Since the distance function satisfies the triangle inequality, i.e., for all nodes u, v, w, $\mathsf{dist}_G(u, v) \leq \mathsf{dist}_G(u, w) + \mathsf{dist}_G(w, v)$, we can express the lower and upper bounds on the graph diameter in terms of the query distance as follows.

LEMMA 2. *For a graph $G(V, E)$ and a set of nodes $Q \subseteq G$, we have $\mathsf{dist}_G(G, Q) \leq \mathsf{diam}(G) \leq 2\mathsf{dist}_G(G, Q)$.*

PROOF. First, the diameter $\mathsf{diam}(G) = \max_{v, u \in G} \mathsf{dist}_G(v, u)$, which is clearly no less than $\mathsf{dist}_G(G, Q) = \max_{v \in G, q \in Q} \mathsf{dist}_G(v, q)$ for $Q \subseteq G$. Thus, $\mathsf{dist}_G(G, Q) \leq \mathsf{diam}(G)$. Second, suppose that the longest shortest path in G is between v and u. Then $\forall q \in Q$, then we have $\mathsf{diam}(G) = \mathsf{dist}(v, u) \leq \mathsf{dist}(v, q) + \mathsf{dist}(q, u) \leq 2\mathsf{dist}_G(G, Q)$. The lemma follows. $\square$

3.2 Free Rider Effect

In previous work on community detection, researchers [25, 28] have identified an undesirable phenomenon called "free rider effect". Intuitively, if a definition of community admits irrelevant subgraphs in the detected community, we refer to such irrelevant subgraphs as free riders. For instance, suppose we use the classic density definition of average internal degree $\frac{|E|}{|V|}$ as the community goodness metric. Then for a set of query nodes Q, the community is a subgraph containing Q with the maximum density. Then, any local community for Q merged with the densest subgraph part will increase the community density. However, the densest subgraph may be disconnected from or irrelevant to query nodes. This shows the simple density metric suffers from the free rider effect. Wu et al. [28] show that several other goodness metrics including minimum degree, local modularity, and external conductance suffer from the free rider effect. Following Wu et al. [28], we define the free rider effect as follows. Typically, a community definition is based on a goodness metric $f(H)$ for a subgraph H: subgraphs with minimum[1] $f(H)$ value are defined as communities. E.g., for our CTC problem, diameter is the goodness metric: among all subgraphs with maximum trussness, the smaller the diameter of H, the better it is as a community. The definition of free rider effect is based on this goodness metric. We term a community query independent if it is the solution to the community search with Q set to $\emptyset$.

DEFINITION 6 (FRE). *Given a non-empty query Q, let H be a solution to a community definition based on a goodness metric $f(.)$. Let H^* be a (global or local) optimum solution, which is query-independent. If $f(H \cup H^*) \leq f(H)$, we say that the definition suffers from free rider effect. Here, nodes in $H^* \setminus H$ are called* free riders *for the query Q and community H.*

EXAMPLE 3. Consider Figure 2, showing a graph G and query nodes $Q = \{q_1, q_2\}$. It also shows subgraphs G_1 and G_2. All three graphs – G, G_1, and G_2 – are 4-trusses containing Q. The query distance of the star node r is 3, while that for all other nodes is at most 2. Thus, the query distance of G is 3. The subgraph G_1 has

---
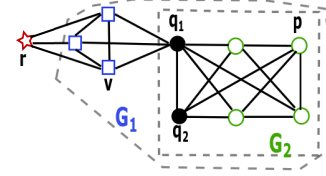[1] We use minimum w.l.o.g.

Figure 2: A graph G with $Q = \{q_1, q_2\}$.

the minimum query distance 2 among all 4-trusses containing Q. However, its diameter is 3, as the distance between square node v and circle node p is 3. On the other hand, the subgraph G_2, while having the same query distance as G_1, has a strictly smaller diameter 2. It has the minimum diameter among all 4-trusses containing Q.

Both the star node and the square nodes are free riders. The star node is the furthest from query node q_2 and its removal from G leaves the trussness unchanged. The square nodes have the same query distance 2 as the circle node p. However, the square nodes are not close enough to *other* nodes of the community: e.g., their distance to circle node p is 3. Unlike the circle nodes, removal of square nodes leaves the trussness unchanged. Thus, the square nodes are also free riders, while the circle nodes aren't. Minimizing query distance among 4-trusses eliminates the free rider star node but not the square free rider nodes, while minimizing diameter eliminates both free riders. $\square$

We next show that our definition of CTC avoids the problem of free rider effect.

In general, there may be multiple CTCs H, i.e., connected k-trusses with maximum trussness containing Q with the minimum diameter. For example, consider the graph G in Figure 1 and $Q = \{q_3\}$. The subgraphs of G induced respectively by $\{q_3, p_1, p_2, p_3\}$ and $\{q_3, v_3, v_4, v_5\}$ are both 4-trusses with diameter 1. Both happen to be maximal in that they are not contained in any other 4-truss with this property.

PROPOSITION 1. *For any graph G and query nodes $Q \subset V(G)$, there is a solution H to the CTC search problem such that for all query-independent optimal solutions H^*, either $H^* = H$, or $H \cup H^*$ is disconnected, or $H \cup H^*$ has a strictly larger diameter than H.*

PROOF. Let $\mathcal{C}(G, Q)$ denote the set of optimal solutions to the CTC search problem on graph G and query nodes Q. $\mathcal{C}(G, Q)$ is partially ordered w.r.t. the graph containment order $\subseteq$. Let H be any maximal element of $\mathcal{C}(G, Q)$, let H^* be any query-independent optimal solution, and consider $H \cup H^*$. Assume w.l.o.g. that $(H^* \setminus H) \neq \emptyset$. Suppose that $H \cup H^*$ is a connected k-truss with maximum trussness containing Q, and $\mathsf{diam}(H \cup H^*) \leq \mathsf{diam}(H)$. This contradicts the maximality of H. $\square$

3.3 Hardness and Approximation

Hardness. In the following, we show the CTC-Problem is NP-hard. Thereto, we define the decision version of the CTC-Problem.

PROBLEM 2 (CTCk-Problem). *Given a graph $G(V, E)$, a set of query nodes $Q = \{v_1, ..., v_r\} \subseteq V$ and parameters k and d, test whether G contains a connected k-truss subgraph with diameter at most d, that contains Q.*

THEOREM 1. *The CTCk-Problem is **NP-hard**.*

PROOF. We reduce the well-known NP-hard problem of Maximum Clique (decision version) to CTCk-Problem. Given a graph $G(V, E)$ and number k, the Maximum Clique Decision problem is

to check whether G contains a clique of size k. From this, construct an instance of CTCk-Problem, consisting of graph G, parameters k and $d = 1$, and the empty set of query nodes $Q = \emptyset$. We show that the instance of the Maximum Clique Decision problem is a YES-instance iff the corresponding instance of CTCk-Problem is a YES-instance. Clearly, any clique with at least k nodes is a connected k-truss with diameter 1. On the other hand, given a solution H for CTCk-Problem, H must contain at least k nodes since H is a k-truss, and $diam(H) = d = 1$, which implies H is a clique. □

The hardness of CTC-Problem follows from this. The next natural question is whether CTC-Problem can be approximated.

Approximation. For $\alpha \geq 1$, we say that an algorithm achieves an α-approximation to the closest truss community (CTC) search problem if it outputs a connected k-truss subgraph $H \subseteq G$ such that $Q \subseteq H$, $\tau(H) = \tau(H^*)$ and $\mathsf{diam}(H) \leq \alpha \cdot \mathsf{diam}(H^*)$, where H^* is the optimal CTC. That is, H^* is a connected k-truss with the largest k s.t. $Q \subseteq H^*$, and $diam(H^*)$ is the minimum among all such CTCs containing Q. Notice that the trussness of the output subgraph H matches that of the optimal solution H^* and that the approximation is only w.r.t. the diameter: the diameter of H is required to be no more than $\alpha \cdot diam(H^*)$.

Non-Approximability. We next prove that CTC-Problem cannot be approximated within a factor better than 2. We establish this result through a reduction, again from the Maximum Clique Decision problem to the problem of approximating CTC-Problem, given k. In the next section, we develop a 2-approximation algorithm for CTC-Problem, thus essentially matching this lower bound. Notice that the CTC-Problem with given parameter k is essentially the CTCk-Problem.

THEOREM 2. *Unless* P = NP, *for any* $\varepsilon > 0$, *the* CTC-Problem *with given parameter* k *cannot be approximated in polynomial time within a factor* $(2 - \varepsilon)$ *of the optimal.*

Proof Sketch: We can show that a $(2 - \varepsilon)$-approximation algorithm for the CTC-Problem with given parameter k can be used to distinguish between the YES and NO instances of the Maximum Clique Decision problem. □

4. ALGORITHMS

In this section, we present a greedy algorithm called Basic for the CTC search problem. Then, we show that this algorithm achieves a 2-approximation to the optimal result. Finally, we discuss procedures for an efficient implementation of the algorithm and analyze its time and space complexity.

4.1 Basic Algorithmic Framework

Here is an overview of our algorithm Basic. First, given a graph G and query nodes Q, we find a maximal connected k-truss, denoted as G_0, containing Q and having the largest trussness. As G_0 may have a large diameter, we iteratively remove nodes far away from the query nodes, while maintaining the trussness of the remainder graph at k.

Algorithm. Algorithm 1 outlines a framework for finding a closest truss community based on a greedy strategy. For query nodes Q, we first find a maximal connected k-truss G_0 that contains Q, s.t. $k = \tau(G_0)$ is the largest (line 1). Then, we set $l = 0$. For all $u \in G_l$ and $q \in Q$, we compute the shortest distance between u and q (line 4), and obtain the vertex query distance $\mathsf{dist}_{G_l}(u, Q)$. Among all vertices, we pick up a vertex u^* with the maximum $\mathsf{dist}_{G_l}(u^*, Q)$, which is also the graph query distance $\mathsf{dist}_{G_l}(G_l, Q)$ (lines 5-6).

Algorithm 1 Basic (G, Q)

Input: A graph $G = (V, E)$, a set of query nodes $Q = \{q_1, ..., q_r\}$.
Output: A connected k-truss R with a small diameter.

1: Find a maximal connected k-truss containing Q with the largest k as G_0; //see Algorithm 2
2: $l \leftarrow 0$;
3: **while** $\mathsf{connect}_{G_l}(Q) = $ **true do**
4:     Compute $\mathsf{dist}_{G_l}(q, u)$, $\forall q \in Q$ and $\forall u \in G_l$;
5:     $u^* \leftarrow \arg\max_{u \in G_l} \mathsf{dist}_{G_l}(u, Q)$;
6:     $\mathsf{dist}_{G_l}(G_l, Q) \leftarrow \mathsf{dist}_{G_l}(u^*, Q)$;
7:     Delete u^* and its incident edges from G_l;
8:     Maintain k-truss property of G_l; //see Algorithm 3
9:     $G_{l+1} \leftarrow G_l; l \leftarrow l + 1$;
10: $R \leftarrow \arg\min_{G' \in \{G_0, ..., G_{l-1}\}} \mathsf{dist}_{G'}(G', Q)$;

Next, we remove the vertex u^* and its incident edges from G_l, and delete any nodes and edges needed to restore the k-truss property of G_l (lines 7-8). We assign the updated graph as a new G_l. Then, we repeat the above steps until G_l does not have a connected subgraph containing Q (lines 3-9). Finally, we terminate by outputting graph R as the closest truss community, where R is any graph $G' \in \{G_0, ..., G_{l-1}\}$ with the smallest graph query distance $\mathsf{dist}_{G'}(G', Q)$ (line 10). Note that each intermediate graph $G' \in \{G_0, ..., G_{l-1}\}$ is a k-truss with the maximum trussness as required.

EXAMPLE 4. *We apply Algorithm 1 on G in Figure 1 for $Q = \{q_1, q_2, q_3\}$. First, we obtain the 4-truss subgraph G_0 shaded in grey, using a procedure we will shortly explain. Then, we compute all shortest distances, and get the maximum vertex query distance as $\mathsf{dist}_{G_0}(p_1, Q) = 4$, and $u^* = p_1$. We delete node p_1 and its incident edges from G_0; we also delete p_2 and p_3, in order to restore the 4-truss property. The resulting subgraph is G_1. Any further deletion of a node in the next iteration of the while loop will induce a series of deletions in line 8, eventually making the graph disconnected or containing just a part of query nodes. As a result, the output graph R, shown in Figure 1(b), is just G_1. Also $\mathsf{dist}_R(R, Q) = 3$, and R happens to be the exact CTC with diameter 3, which is optimal.*

4.2 Approximation Analysis

Algorithm 1 can achieve 2-approximation to the optimal solution, that is, the obtained connected k-truss community R satisfies $Q \subseteq R$, $\tau(R) = \tau(H^*)$ and $\mathsf{diam}(R) \leq 2\mathsf{diam}(H^*)$, for any optimal solution H^*. Since any graph in $\{G_0, ..., G_{l-1}\}$ is a connected k-truss with the largest k containing Q by Algorithm 1, and $R \in \{G_0, ..., G_{l-1}\}$, we have $Q \subseteq R$, and $\tau(R) = \tau(H^*)$. In the following, we will prove that $\mathsf{diam}(R) \leq 2\mathsf{diam}(H^*)$. We start with a few key results. For graphs G_1, G_2, we write $G_1 \subseteq G_2$ to mean $V(G_1) \subseteq V(G_2)$ and $E(G_1) \subseteq E(G_2)$.

FACT 1. *Given two graphs G_1 and G_2 with $G_1 \subseteq G_2$, for $u, v \in V(G_1)$, $\mathsf{dist}_{G_2}(u, v) \leq \mathsf{dist}_{G_1}(u, v)$ holds. Moreover, if $Q \subseteq V(G_1)$, then $\mathsf{dist}_{G_2}(G_1, Q) \leq \mathsf{dist}_{G_1}(G_1, Q)$ also holds.*

PROOF. Trivially follows from the fact that G_2 preserves paths between nodes in G_1. □

Recall that in Algorithm 1, in each iteration i, a node u^* with maximum $\mathsf{dist}(u^*, Q)$ is deleted from G_i, but $\mathsf{dist}_{G_i}(G_i, Q)$ is *not* monotone nonincreasing during the process, hence $\mathsf{dist}_{G_{l-1}}(G_{l-1}, Q)$ is not necessarily the minimum. Note that in Algorithm 1, G_l is not the last feasible graph (i.e., connected k-truss containing Q), but G_{l-1} is. The observation is shown in the following lemma.

LEMMA 3. *In Algorithm 1, it is possible that for some $0 \leq i < j < l$, we have $G_j \subset G_i$, and $\mathsf{dist}_{G_i}(G_i, Q) < \mathsf{dist}_{G_j}(G_j, Q)$ hold.*

PROOF. It is easy to be realized, because for a vertex $v \in G$, $\mathsf{dist}_G(v, Q)$ is non-decreasing monotone w.r.t. subgraphs of G. More precisely, for $v \in G_i \cap G_j$, $\mathsf{dist}_{G_i}(v, Q) \leq \mathsf{dist}_{G_j}(v, Q)$ holds. $\square$

We have an important observation that if an intermediate graph G_i obtained by Algorithm 1 contains an optimal solution H^*, i.e., $H^* \subset G_i$, and $\mathsf{dist}_{G_i}(G_i, Q) > \mathsf{dist}_{G_i}(H^*, Q)$, then algorithm will not terminate at G_{i+1}.

LEMMA 4. *In Algorithm 1, for any intermediate graph G_i, we have $H^* \subseteq G_i$, and $\mathsf{dist}_{G_i}(G_i, Q) > \mathsf{dist}_{G_i}(H^*, Q)$, then G_{i+1} is a connected k-truss containing Q and $H^* \subseteq G_{i+1}$.*

PROOF. Suppose $H^* \subseteq G_i$ and $\mathsf{dist}_{G_i}(G_i, Q) > \mathsf{dist}_{G_i}(H^*, Q)$. Then there exists a node $u \in G_i \backslash H^*$ s.t. $\mathsf{dist}_{G_i}(u, Q) = \mathsf{dist}_{G_i}(G_i, Q) > \mathsf{dist}_{G_i}(H^*, Q)$. Clearly, $u \notin Q$. In the next iteration, Algorithm 1 will delete u from G_i (Step 7), and perform Step 8. The graph resulting from restoring the k-truss property is G_{i+1}. Since H^* is a connected k-truss containing Q, the restoration step (line 8) must find a subgraph G_{i+1} s.t. $H^* \subseteq G_{i+1}$, and G_{i+1} is a connected k-truss containing Q. Thus, the algorithm will not terminate in iteration $(i + 1)$. $\square$

We are ready to establish the main result of this section. Our polynomial algorithm can find a connected k-truss community R having the minimum query distance to Q, which is optimal.

LEMMA 5. *For any connected k-truss H with the highest k containing Q, $\mathsf{dist}_R(R, Q) \leq \mathsf{dist}_H(H, Q)$.*

PROOF. The following cases arise for G_{l-1}, which is the last feasible graph obtained by Algorithm 1.

Case (a): $H \subseteq G_{l-1}$. We have $\mathsf{dist}_{G_{l-1}}(G_{l-1}, Q) \leq \mathsf{dist}_{G_{l-1}}(H, Q)$; for otherwise, if $\mathsf{dist}_{G_{l-1}}(G_{l-1}, Q) > \mathsf{dist}_{G_{l-1}}(H, Q)$, we can deduce from Lemma 4 that G_{l-1} is not the last feasible graph obtained by Algorithm 1, a contradiction. Thus, by Step 10 in Algorithm 1 and the fact that $\mathsf{dist}_{G_{l-1}}(G_{l-1}, Q) \leq \mathsf{dist}_{G_{l-1}}(H, Q)$, we have $\mathsf{dist}_R(R, Q) \leq \mathsf{dist}_{G_{l-1}}(G_{l-1}, Q) \leq \mathsf{dist}_{G_{l-1}}(H, Q) \leq \mathsf{dist}_H(H, Q)$.

Case (b): $H \nsubseteq G_{l-1}$. There exists a vertex $v \in H$ deleted from one of the subgraphs $\{G_0, ..., G_{l-2}\}$. Suppose the first deleted vertex $v^* \in H$ is in graph G_i, where $0 \leq i \leq l - 2$, then v^* must be deleted in Step 7, but not in Step 8. This is because each vertex/edge of H satisfies the condition of k-truss, and will not be removed before any vertex is removed from G_i. Then, we have $\mathsf{dist}_{G_i}(G_i, Q) = \mathsf{dist}_{G_i}(v^*, Q) = \mathsf{dist}_{G_i}(H, Q)$, and $\mathsf{dist}_{G_i}(G_i, Q) \geq \mathsf{dist}_R(R, Q)$ by Step 10. As a result, $\mathsf{dist}_R(R, Q) \leq \mathsf{dist}_{G_i}(H, Q) \leq \mathsf{dist}_H(H, Q)$. $\square$

Based on the preceding lemmas, we have:

THEOREM 3. *Algorithm 1 provides a 2-approximation to the* CTC-Problem *as $diam(R) \leq 2diam(H^*)$.*

PROOF. Since $\mathsf{dist}_R(R, Q) \leq \mathsf{dist}_{H^*}(H^*, Q)$ by Lemma 5, we get $diam(R) \leq 2\mathsf{dist}_R(R, Q) \leq 2\mathsf{dist}_{H^*}(H^*, Q) \leq 2diam(H^*)$ by Lemma 2. The theorem follows from this. $\square$

4.3 K-truss Identification and Maintenance

In this section, we introduce the detailed implementation of Algorithm 1. Finding G_0, the maximal connected k-truss containing Q with the largest trussness k, is a basic primitive in our problem.

Algorithm 2 Find$G_0(G, Q)$

Input: A graph $G = (V, E)$, a set of query nodes $Q = \{q_1, ..., q_r\}$.
Output: A connected k-truss G_0 containing Q with the largest k.

1: $k \leftarrow \min \{\tau(q_1), ..., \tau(q_r)\}$ //see Lemma 1;
2: $V(G_0) \leftarrow \emptyset; S_k = Q$;
3: **while** connect$_{G_0}(Q) =$ **false do**
4:      **for** $v \in S_k$ **do**
5:          **if** $v \in V(G_0)$ **then**
6:              $k_{max} \leftarrow k + 1$;
7:          **else**
8:              $k_{max} \leftarrow +\infty; V(G_0) \leftarrow V(G_0) \cup \{v\}$;
9:          **for** $(v, u) \in G$ with $k \leq \tau(v, u) < k_{max}$ **do**
10:             $G_0 \leftarrow G_0 \cup \{(v, u)\}$;
11:             **if** $u \notin S_k$ **then** $S_k \leftarrow S_k \cup \{u\}$;
12:          $l \leftarrow \max\{\tau(v, u) | (v, u) \notin G_0\}$;
13:          $S_l \leftarrow S_l \cup \{v\}$;
14:      $k \leftarrow k - 1$;
15: Compute the edge support $sup(v, u)$ in G_0, for all $(v, u) \in G_0$;

A straightforward method is to apply a truss decomposition algorithm [27], and delete edges in ascending order of edge support from G, until Q becomes disconnected. Then we can obtain the largest trussness k and recover G_0 by keeping all k-truss edges. However, this method is quite costly. To find G_0 efficiently, we design an index structure. The index is constructed by organizing edges according to their trussness.

Index Construction. We first apply a truss decomposition algorithm such as [27] and compute the trussness of each edge of graph G. We omit the details of this algorithm due to space limitation.

Based on the obtained edge trussness, we construct our truss index as follows. For each vertex $v \in V$, we sort its neighbors $N(v)$ in descending order of the edge trussness $\tau(e(v, u))$, for $u \in N(v)$. For each distinct trussness value $k \geq 2$, we mark the position of the first vertex u in the sorted adjacency list where $\tau(e(u, v)) = k$. This supports efficient retrieval of v's incident edges with a certain trussness value. The vertex trussness of v is also kept as $\tau(v) = \max\{\tau(v, u) | u \in N(v)\}$, which is the trussness of the first edge in the sorted adjacency list. Moreover, we build a hashtable to keep all the edges and their trussness values. This is identical to the simple truss index of [16] and we refer to it as the truss index.

In the following, we will show that this truss index is sufficient to design an algorithm for finding the maximal connected k-truss containing given query nodes Q in time $O(m')$, where $m' = |E(G_0)|$. This time complexity is essentially *optimal*. We remark the complexity of this k-truss index construction below.

REMARK 1. *The construction of this truss index takes $O(\rho \cdot m)$ time and $O(m)$ space, where ρ is the arboricity of graph G, i.e., the minimum number of spanning forests needed to cover all edges of G. Notice that $\rho \leq \min\{d_{max}, \sqrt{m}\}$ [5].*

Finding G_0. Based on the index, we present Algorithm 2 for finding G_0, the maximal connected k-truss containing Q with the largest trussness k. We initialize G_0 to be the query vertex set Q, and iteratively add the edges of G in decreasing order of trussness, until G_0 gets connected.

The initial trussness level of the edges to be included in G_0 is computed as $k = \min\{\tau(q_1), ..., \tau(q_r)\}$ (line 1). This is motivated by the fact that, by Lemma 1, for any $k' > k$, no connected k'-truss can contain Q. We use S_k to denote the set of nodes to be visited within level k. We start with $S_k = Q$ (line 2). For a given k, we process each node $v \in S_k$, and visit its neighbors in a BFS manner. Then, we insert those of its incident edges
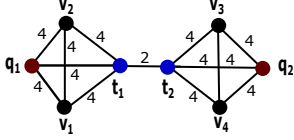
Figure 3: An example graph G of finding G_0

(v, u), with $k \leq \tau(v, u) \leq k_{max}$ into G_0, where k_{max} is the maximum possible trussness of unvisited edges. This is because all these edge should be present in a connected k-truss. Meanwhile, if the neighbor u is not in S_k, we add u into S_k (line 11), since unvisited edges incident to u may have trussness no less than k. After checking all edges incident to v, we add v to S_l, where $l = \max\{\tau(v, u) \mid u \in N(v), \tau(v, u) < k\}$ (line 12-13). Notice that l is the next highest level for which a connected l-truss contains the node v, which can avoid scanning the neighbor set of v at each level. After traversing all vertices in S_k, the algorithm checks whether Q is connected in G_0. If yes, the algorithm terminates, and G_0 is returned; otherwise, we decrease the present level k by 1 (line 14), and repeat the above steps (lines 4-14). After obtaining G_0, we compute all edge supports by counting triangles in G_0, which is used for the k-truss maintenance (line 15).

The following example illustrates the algorithm.

EXAMPLE 5. Consider the graph G in Figure 3 with $Q = \{q_1, q_2\}$. The trussness of each edge is displayed, e.g., $\tau(q_1, v_1) = 4$. Now, we apply Algorithm 2 on G to find G_0 containing Q. We can verify that $\tau(q_1) = \tau(q_2) = 4$ so we start with level $k = 4$ and set $S_4 = \{q_1, q_2\}$. Then, we process the node $q_1 \in S_4$, and insert all its incident edges into G_0, for the trussness of each edge is 4. Meanwhile, all its neighbors are inserted into S_4. We repeat above process for each node in S_4. Note that for nodes $t_1, t_2, \tau(t_1, t_2) = 2$, so we insert t_1, t_2 into S_2 (lines 11-12 of Algorithm 2). Then, at level $k = 4$, we get the 4-truss as the whole graph in Figure 3 minus the edge (t_1, t_2), for $\tau(t_1, t_2) = 2$. Since the current G_0 is not connected, we decrease the truss level k to 3, and find that $S_3 = \emptyset$. Then, we decrease k to 2, and find that $S_2 = \{t_1, t_2\}$. So we expand from the edge incident to t_1, and insert the edge (t_1, t_2) into G_0, and find that the resulting graph contains Q and is connected. In this example, G_0 happens to coincide with G. □

REMARK 2. *Based on the truss index, for each vertex v, in line 9 of Algorithm 2, each edge (v, u) can be accessed constant time using the sorted adjacent list of v, and in line 12, we can compute l in constant time. Algorithm 2 takes time $O(m')$ where $m' = |E(G_0)|$.*

Computing Query Distance. For a vertex v, to compute the query distance $\mathsf{dist}_{G_i}(v, Q)$, we need to perform $|Q|$ BFS traversals on graph G_i. Specifically, for each query node $q \in Q$, with one BFS traversal starting from q in G_i, we can obtain the shortest distance $\mathsf{dist}_{G_i}(v, q)$ for each node $v \in G_i$. Then, $\mathsf{dist}_{G_i}(v, Q)$ is the maximum of all shortest distances $\mathsf{dist}_{G_i}(v, q)$, for $q \in Q$.

K-truss Maintenance. Algorithm 3 describes the procedure for maintaining G as a k-truss after the deletion of nodes V_d from G. In Algorithm 1, $V_d = \{u^*\}$ (see line 8).[2] Generally speaking, after removing nodes V_d and their incident edges from G, G may not be a k-truss any more, or Q may be disconnected. Thus, Algorithm 3 iteratively deletes edges having support less than $(k-2)$ and nodes disconnected with Q from G, until G becomes a connected k-truss containing Q.

Algorithm 3 firstly pushes all edges incident to nodes V_d into set S (lines 1-3). Then, for each edge $(u, v) \in S$, the algorithm checks

---

[2]In Section 5, we will discuss deleting a set of nodes V_d in batch.

Algorithm 3 K-trussMaintenance (G, V_d)

Input: A graph $G = (V, E)$, a set of nodes to be removed as V_d.
Output: A k-truss graph.

1: $S \leftarrow \emptyset$; //S is the set of removed edges.
2: **for** $v \in V_d$ **and** $(v, u) \in G$ **do**
3: $\quad S \leftarrow S \cup (v, u)$;
4: **for** $(v, u) \in S$ **do**
5: $\quad$ **for** $w \in N(v) \cap N(u)$ **do**
$\quad\quad$ // Update the support of edges (v, w) and (u, w)
6: $\quad\quad sup(v, w) \leftarrow sup(v, w) - 1$; $sup(u, w) \leftarrow sup(u, w) - 1$;
7: $\quad\quad$ **if** $sup(v, w) < k - 2$ **and** $(v, w) \notin S$ **then** $S \leftarrow S \cup (v, w)$;
8: $\quad\quad$ **if** $sup(u, w) < k - 2$ **and** $(u, w) \notin S$ **then** $S \leftarrow S \cup (u, w)$;
9: $\quad$ Remove (v, u) from G;
10: Remove isolated vertices from G;

---

every triangle $\triangle_{uvw}$ where $w \in N(u) \cap N(v)$, and decreases the support of edges (u, w) and (v, w) by 1; For any edge $e \notin S$, with resulting support $sup(e) < k - 2$, e is added to S. After traversing all triangles containing (u, v), the edge (u, v) is deleted from G. This process continues until S becomes empty (lines 4-9), and then the algorithm removes all isolated vertices form G (line 10).

4.4 Complexity analysis

In the implementation of Algorithm 1, we do not need to keep all immediate graphs, but just record the removal of vertices/edges at each iteration. Let G_0 be the maximal connected k-truss found in line 1 of Algorithm 1. Let $n' = |V(G_0)|$ and $m' = |E(G_0)|$, and let d'_{max} be the maximum degree of a vertex in G_0.

At each iteration i of Algorithm 1, we delete at least one node and its incident edges from G_i. Clearly, the number of removed edges is no less than $k - 1$, thus the total number of iterations is $t \leq \min\{n' - k, m'/(k-1)\}$, i.e., t is $O(\min\{n', m'/k\})$. We have:

THEOREM 4. *Algorithm 1 takes $O((|Q|t + \rho)m')$ time and $O(m')$ space, where $t \in O(\min\{n', m'/k\})$, and ρ is the arboricity of graph G_0. Furthermore, we have $\rho \leq \min\{d'_{max}, \sqrt{m'}\}$.*

Proof Sketch: First, listing all triangles of G_0 and creating a series of k-truss graphs $\{G_0, ..., G_{l-1}\}$ take $O(\rho m')$ time in all, where ρ is the arboricity of graph G_0. Second, the computation of shortest distances by a BFS traversal starting from each query node $q \in Q$ takes $O(t|Q|m')$ time for t iterations. Third, for the space consumption, we only record the sequence of removed edges from G_0 for attaching a corresponding label to a graph G_i at each iteration i, which takes $O(m')$ space in all. A complete proof is available in an arXiv report [17].

5. FAST SEARCH ALGORITHMS

In this section, we focus on improving the efficiency of CTC search in two ways. First, we develop a new greedy strategy to speed up the pruning process process in Section 5.1, by deleting at least k nodes in batch, to achieve quick termination while sacrificing some approximation ratio. Second, we also propose a heuristic strategy to quickly find the closest truss community in the local neighborhood of query nodes.

5.1 Bulk Deletion Optimization

In this subsection, we propose a new algorithm called BulkDelete following the framework of Algorithm 1, which is based on deletion of a set of nodes in batch when maintaining a k-truss. The algorithm is described in detail in Algorithm 4, which can terminate quicker than Algorithm 1. It is based on the following two observations.

Algorithm 4 BulkDelete (G, Q)

Input: A graph $G = (V, E)$, a set of query nodes $Q = \{q_1, ..., q_r\}$.
Output: A connected k-truss R with a small diameter.

1: Find$G0$ (G, Q) //see Algorithm 2;
2: $d \leftarrow +\infty; l \leftarrow 0$;
3: **while** connect$_{G_l}(Q) = $ **true do**
4:     Compute $\mathsf{dist}_{G_l}(q, u), \forall q \in Q$ and $\forall u \in G_l$;
5:     $\mathsf{dist}_{G_l}(G_l, Q) \leftarrow \max_{u^* \in G_l} \mathsf{dist}_{G_l}(u^*, Q)$;
6:     **if** $\mathsf{dist}_{G_l}(G_l, Q) < d$ **then**
7:         $d \leftarrow \mathsf{dist}_{G_l}(G_l, Q)$;
8:     $L = \{u^* | \mathsf{dist}_{G_l}(u^*, Q) \geq d - 1, u^* \in G_l\}$;
9:     Maintain k-truss property of G_l //see Algorithm 3;
10:    $G_{l+1} \leftarrow G_l; l \leftarrow l + 1$;
11: $R \leftarrow \arg\min_{G' \in \{G_0,...,G_{l-1}\}} \mathsf{dist}_{G'}(G', Q)$;

First, in Algorithm 1, if a graph G_i has query distance $\mathsf{dist}_{G_i}(G_i, Q) = d$, only one vertex u^* with $\mathsf{dist}_{G_i}(u^*, Q) = d$ is removed from G_i. Instead, we can delete *all* nodes u with $\mathsf{dist}_{G_i}(u, Q) = d$, from G_i, in one shot. The reason is that $\mathsf{dist}_{G_i}(u, Q)$ is monotone non-decreasing with decreasing graphs, i.e., $dist_{G_j}(u, Q) \geq \mathsf{dist}_{G_i}(u, Q) = d$, for $j > i$. Thus, removing a set of vertices $L = \{u^* | \mathsf{dist}_{G_i}(u^*, Q) \geq d, u^* \in G_i\}$ in each iteration i will improve the efficiency. This improvement indeed works in real applications. However, in theory, it is possible that $|L| = 1$ in every iteration.

Our second observation, is that a vertex u^* with $\mathsf{dist}_{G_i}(u^*, Q) = d$ has at least $k - 1$ neighbors v with $\mathsf{dist}_{G_i}(v, Q) = d - 1$. If we remove $L = \{u | \mathsf{dist}_{G_i}(u, Q) \geq d - 1, u \in G_i\}$ at each iteration, then at least k nodes are removed each time. Thus, the resulting number of iterations is $O(n'/k)$, where $n' = |V(G_0)|$.

Thus, the number of iterations is improved from $O(\min\{n', m'/k\})$ to $O(n'/k)$ (see Theorem 4). We just proved:

THEOREM 5. *Algorithm 4 takes $O((|Q|t' + \rho')m')$ time using $O(m')$ space, where $t' \in O(n'/k)$, and $\rho' \leq \min\{d'_{max}, \sqrt{m'}\}$.*

The approximation quality of Algorithm 4 is characterized below.

THEOREM 6. *Algorithm 4 is a $(2 + \varepsilon)$-approximation solution of CTC-Problem, where $\varepsilon = 2/diam(H^*)$.*

PROOF. To prove this theorem, we only need to ensure $\mathsf{dist}_R(R, Q) \leq \mathsf{dist}_{H^*}(H^*, Q) + 1$. Because $diam(R) \leq 2\mathsf{dist}_R(R, Q) \leq 2\mathsf{dist}_{H^*}(H^*, Q) + 2 \leq 2(diam(H^*) + 1)$ by Lemma 2, then approximation ratio is $2 + \varepsilon$, where $\varepsilon = 2/diam(H^*)$. The detailed proof is similar with Lemma 5, which is omitted here, due to space limitation. □

EXAMPLE 6. Continuing with the previous example, we apply Algorithm 4 on Figure 1(a) to find the closest truss community for $Q = \{q_1, q_2, q_3\}$. In G_0, we compute $d = \max_{u \in G_0} \mathsf{dist}_{G_0}(u, Q) = 4$, and $L = \{q_1, q_3, p_1, p_2, p_3\}$, as each node $u \in L$ has query distance $\mathsf{dist}_{G_0}(u, Q) = 3 \geq d - 1$. After removing L from G_0, the remaining graph does not contain Q, and the algorithm terminates. Thus, Algorithm 4 reports the entire 4-truss G_0 as the answer, which has diameter 4, compared to the answer of Figure 1(b) reported by Algorithm 1, which has diameter 3. □

5.2 Local Exploration

In this subsection, we develop a heuristic strategy to quickly find the closest truss community by local exploration. The key idea is as follows. We first form a Steiner tree to connect all query nodes,

---

Algorithm 5 Local-CTC (G, Q)

Input: A graph $G = (V, E)$, a set of query nodes $Q = \{q_1, ..., q_r\}$, a node size threshold η.
Output: A connected k-truss R with a small diameter.

1: Compute a Steiner Tree T containing Q using truss distance functions;
2: $k_t \leftarrow \min_{e \in T} \tau(e)$;
3: Expand T to a graph $G_t = \{e \in G | \tau(e) \geq k_t\}$, s.t. $T \subseteq G_t$ and $|G_t| \leq \eta$;
4: Extract the maximal connected k-truss H_t containing Q from G_t, where $k \leq k_t$ is the maximum possible trussness;
5: Apply a variant of BulkDelete algorithm on H_t to identify closest community.

---

and then expand it to a graph G'_0 by involving the local neighborhood of the query nodes. From this new graph G'_0, we find a connected k-truss with the highest k containing Q, and then iteratively remove the furthest nodes from this k-truss using the BulkDelete algorithm discussed earlier.

Connect query nodes with a Steiner tree. As explained above, the Steiner tree found is used as a seed for expanding into a k-truss. It is well-known that finding a minimal weight Steiner tree is NP-hard but it admits a 2-approximation [18, 21]. However, a naive application of these algorithms may produce a result with a small trussness. To see this, consider the graph G and the query $Q = \{q_1, q_2, q_3\}$ in Figure 1(a). Suppose all edges are uniformly weighted. Then it is obvious that the tree $T_1 = \{(q_2, q_1), (q_1, t), (t, q_3)\}$ with total weight 3 is an optimal (i.e., minimum weight) Steiner tree for Q. However, the smallest trussness of the edges in T_1 is 2, which suggests growing T_1 into a larger graph will yield a low trussness. By contrast, the Steiner tree $T_2\{(q_1, q_2), (q_2, v_4), (v_4, q_3)\}$ has the total weight 3 and all its edges have the trussness at least 4, indicating it could be expanded into a more dense graph. To help discriminate between such Steiner trees, we define path weights as follows. Recall the definition of $\bar{\tau}(S)$ from Section 2.

DEFINITION 7 (TRUSS DISTANCE). *Given a path P between nodes u, v in G, we define the truss distance of u and v as $\hat{\mathsf{dist}}_P(u, v) = \mathsf{dist}_P(u, v) + \gamma(\bar{\tau}(\emptyset) - \min_{e \in P} \tau(e))$, where $\mathsf{dist}_P(u, v)$ is the path length of P, and $\gamma > 0$. For a tree T, by $\hat{\mathsf{dist}}_T(u, v)$ we mean $\hat{\mathsf{dist}}_P(u, v)$ where P is the path connecting u and v in T.*

The difference $\bar{\tau}(\emptyset) - \min_{e \in P} \tau(e)$ measures how much the minimum edge trussness of path P falls short of the maximum edge trussness of the graph G and γ controls the extent to which small edge trussness is penalized. The larger γ is, the more important edge trussness is in distance calculations. Note that, for the special case when P consists of a single edge (u, v), the minimum edge truss in P is just $\tau(u, v)$. On the other hand, for a path P of length more than 1, the penalty only depends on the minimum edge trussness of path P, and does not account for every edge in P. In order to leverage the well-known approximation algorithm of Steiner tree algorithm [21], we define the truss distance for a path. Recall the procedure of Steiner tree algorithm [21], given a graph G and query nodes Q, it firstly constructs a complete distance graph G' of query nodes where the distance equals its shortest path length in G. It then finds a minimum spanning tree T of G', and then constructs another graph H by replacing each edge of tree T by its corresponding shortest path in G, and finally finds a minimum spanning tree of H and deletes leaf edges. We apply the truss distance function on the path weight for shortest path and minimum spanning tree construction here. For instance, in the above example, $\bar{\tau}(\emptyset) = 4$ and for $\gamma = 3$, the truss distance of (q_2, q_3) in T_1 is $\hat{\mathsf{dist}}_{T_1}(q_2, q_3) = \mathsf{dist}_{T_1}(q_2, q_3) + 3 \cdot (4 - 2) = 3 + 6 = 8$, since

Table 2: Network statistics (K $= 10^3$ and M $= 10^6$)

Network	$|V_G|$	$|E_G|$	d_{max}	$\bar{\tau}(\emptyset)$
Facebook	4**K**	88**K**	1,045	97
Amazon	335**K**	926**K**	549	7
DBLP	317**K**	1**M**	342	114
Youtube	1.1**M**	3 **M**	28,754	19
LiveJournal	4**M**	35**M**	14,815	352
Orkut	3.1**M**	117**M**	33,313	78

the minimum edge trussness of T_1 is $\tau(q_1, t) = 2$. On the other hand, $\hat{\text{dist}}_{T_2}(q_1, q_3) = \text{dist}_{T_2}(q_1, q_3) + 3 \cdot (4 - 4) = 3 + 0 = 3$. Obviously, the Steiner tree T_2 has a smaller truss distances than T_1. It can be verified that its overall weight is smaller than that of T_1.

Find G_0 by expanding Steiner tree to graph. After obtaining the Steiner tree T for the query nodes, we locally expand the tree to a small graph G_t as follows. We firstly obtain the minimum trussness of edges in T as $k_t = \min_{e \in T} \tau(e)$. Then, we start from the nodes in T, and expand the tree to a graph in a BFS manner via edges of trussness no less than k_t, and iteratively insert these edges into G_t until the node size exceeds a threshold η, i.e., $|V(G_t)| \leq \eta$, were η is empirically tuned. Since G_t is a local expansion of T, the trussness of G_t will be at most k_t, i.e., $\tau(G_t) \leq k_t$. For ensuring the dense cohesive structure of identified communities, we apply a truss decomposition algorithm on G_t. Then, we extract the maximal connected k-truss subgraph H_t containing Q by removing all edges of trussness less than k from G_t, where $k \leq k_t$ is the maximum possible trussness.
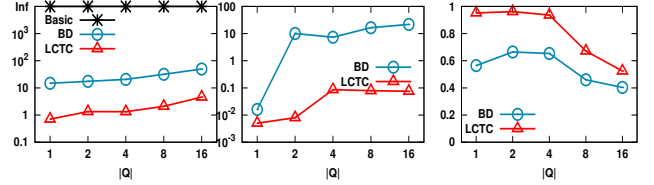
Reduce the diameter of G_0. We take the graph H_t with the maximum trussness k as input, and apply a variant of BulkDelete algorithm on H_t for returning the identified community. We implement a variant of BulkDelete algorithm, which is different from original BulkDelete w.r.t. the removed vertex set $L = \{u^* | \text{dist}_{G_l}(u^*, Q) \geq d - 1, u^* \in G_l\}$. We readjust the furthest nodes to be removed, as $L' = \{u^* | \text{dist}_{G_l}(u^*, Q) \geq d, u^* \in G_l\}$. This adjustment makes the algorithm not as efficient as BulkDelete in asymptotic running time complexity, but we still find it efficient in practice. On the other hand, in practice, this strategy can achieve a smaller graph diameter than BulkDelete. Moreover, in our implementation, in each iteration, we carefully remove only a subset of nodes in L', which have the largest total of distances from all query nodes. As a result, more nodes with the largest query distance are removed from the community in the end. The reason is as follows. Suppose the largest query distance we found as d, in the real world, the number of nodes having query distance d may be large, due to the small-world property.
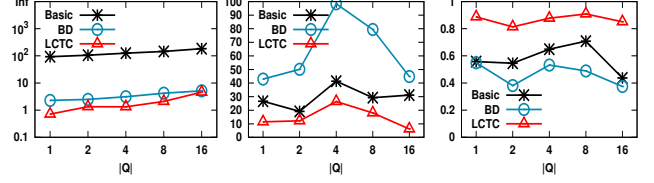
6. EXPERIMENTS

We conduct experimental studies using 6 real-world networks available from the Stanford Network Analysis Project[3], where all networks are treated as undirected. The network statistics are shown in Table 2. All networks except for Facebook contain 5,000 top-quality ground-truth communities.

To evaluate the efficiency and effectiveness of improved strategies, we test and compare three algorithms proposed in this paper, namely, Basic, BD, and LCTC. Here, Basic is the basic greedy approach Basic in Algorithm 1, which removes single furthermost node at each iteration. BD is the BulkDelete approach in Algorithm 4, which removes multiple furthermost nodes at each iteration. LCTC is the local exploration approach in Algorithm 5. For LCTC, we set the parameters $\eta = 1,000$ and $\gamma = 3$, where $\eta = 1,000$ is selected to achieve stable quality and efficiency by testing η in $[500, 2, 000]$, and $\gamma = 3$ is selected to balance the requirements of trussness and diameter for communities searched.

[3]snap.stanford.edu

(a) Query Time (b) The percentage (c) Density
Figure 4: DBLP: varying query size $|Q|$

(a) Query Time (b) The percentage (c) Density
Figure 5: Facebook: varying query size $|Q|$

We randomly generate sets of query nodes to test. Three parameters, query size $|Q|$, degree rank Q_d, and inter-distance l, are varied for generating different sets of query nodes. Here, $|Q|$ is the number of query nodes, which is set to 3 by default. Q_d is the degree rank of query nodes. We sort all vertices in descending order of their degrees in a network. A node is said to be with degree rank of X%, if it has top highest X% degree in the network. The default value of Q_d is 80%, which means that a query node has degree higher than the degree of 20% nodes in the whole network. The inter-distance l is the inter-distance between all query nodes. The default $l = 2$ indicates that all query nodes are within distance of 2 to each other in the network.
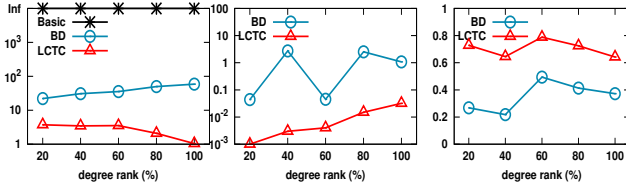
For the efficiency, we report running time in seconds. We treat the running time of a query as infinite if it exceeds 1 hour.

For the effectiveness of eliminating "free riders", we compare our methods with Truss (Algorithm 2), which finds the connected k-truss graph containing query nodes with the largest k only.
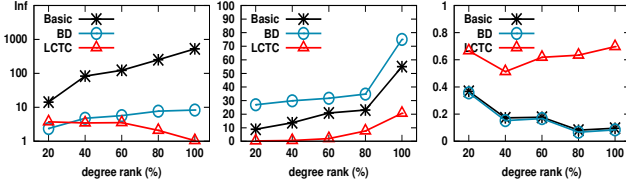
Let G_R be the closest truss community found by LCTC and G_0 be computed by Truss. We report two things. One is the percentage of nodes that are kept in the resulting community by $\frac{|V(G_R)|}{|V(G_0)|}$. The less percentage the more "free riders" being removed. The other is the edge density $2|E(g)|/|V(g)|(|V(g)| - 1)$, where g is either G_R or G_0. Clearly, larger the edge density the more desirable the community.

In addition, to evaluate the quality of closest truss community found, we implemented two state-of-the-art community search methods: the minimum degree-based community search (MDC) [25], which globally finds the dense subgraph containing all query nodes with the highest minimum degree under the distance and size constraints, and the query biased densest community search (QDC) [28], which shifts the detected community to the neighborhood of the query by integrating the edge density and nodes proximity to the query nodes. Here, MDC and QDC are implemented using the same data structures for the graph, Steiner tree, and hashtable as we do for LCTC. To compare LCTC with MDC and QDC, we test the datasets with ground-truth, and show the F1-score to measure the alignment between a discovered community C and a ground-truth community $\hat{C}$. Here, $F1$ is defined as $F1(C, \hat{C}) = \frac{2 \cdot prec(C, \hat{C}) \cdot recall(C, \hat{C})}{prec(C, \hat{C}) + recall(C, \hat{C})}$ where $prec(C, \hat{C}) = \frac{|C \cap \hat{C}|}{|C|}$ is the precision and $recall(C, \hat{C}) = \frac{|C \cap \hat{C}|}{|\hat{C}|}$ is the recall.

All algorithms are implemented in C++, and all the experiments are conducted on a Linux Server with Intel Xeon CUP X5570 (2.93 GHz) and 50GB main memory.

(a) Query Time   (b) The percentage   (c) Density
Figure 6: DBLP: varying query vertices

(a) Query Time   (b) The percentage   (c) Density
Figure 8: DBLP: varying inner distance l

(a) Query Time   (b) The percentage   (c) Density
Figure 7: Facebook: varying query vertices

(a) Query Time   (b) The percentage   (c) Density
Figure 9: Facebook: varying inner distance l

Exp-1 Different Queries: We test our approaches using different queries on DBLP and Facebook (Table 2).

First, we vary the query size $|Q|$. We test 5 different $|Q|$ in $\{1, 2, 4, 8, 16\}$. For each value of $|Q|$, we randomly select 100 sets of $|Q|$ query nodes, and we report the average running time, the average percentage of FRE avoidance and the average edge density.[4] The results for DBLP and Facebook are shown in Figure 4 and Figure 5, respectively. LCTC performs the best in terms of efficiency, the percentage of FRE avoidance, and edge density in all cases. Basic cannot find communities in DBLP within a 1 hour limit we had imposed. BD achieves better efficiency in Facebook than DBLP. This is because Facebook contains only 4K vertices and the global method BD is effective on such a small network. However, BD performs worse than Basic on the percentage of FRE avoidance and edge density for Facebook.

Second, we vary the degree of query nodes. For a graph to be tested, we sort the vertices in descending order of their degrees, and partition them into 5 equal-sized buckets. For each bucket, we randomly select 100 different query sets of size 3, and report the average running time, the average percentage of FRE avoidance and the average edge density. The results for DBLP and Facebook are shown in Figure 6 and Figure 7, respectively. In terms of running time, percentage of FRE avoidance, and edge density, the performance is similar to the results obtained by varying the query sizes. LCTC outperforms all other methods.
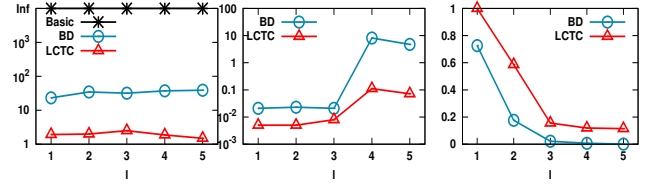
Third, we vary the inter-distance l within query nodes from 1 to 5. For each l value, we randomly select 100 sets of 3 query nodes, in which the inter-distance of query nodes is to be l. We report the average running time, average percentage of FRE avoidance, and edge density. The results for DBLP and Facebook are shown in Figure 8 and Figure 9, respectively. The performance in terms of running time, percentage of FRE avoidance, and edge density is similar to previous results. All methods increase the percentage while the inter-distance l increases. This is because the diameter of community increases, and therefore the less number of nodes can be removed from graph. LCTC outperforms the others.

Exp-2 A Case Study on DBLP: We construct a collaboration network from the raw DBLP data set[5] for a case study. A vertex represents an author, and an edge between two authors indicates they have co-authored no less than 3 times. This DBLP graph contains 234,879 vertices and 541,814 edges.
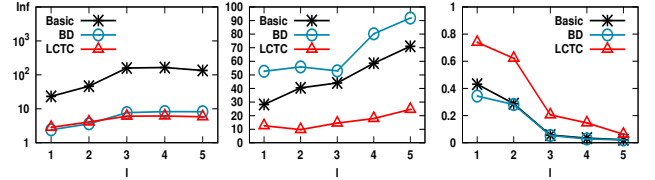
We use the query $Q = \{$"Alon Y. Halevy", "Michael J. Franklin", "Jeffrey D. Ullman", "Jennifer Widom"$\}$ to test our closest truss

---

[4]Notice that Basic, BD, LCTC are not optimal algorithms.
[5]http://dblp.uni-trier.de/xml/

community model for detecting the community. Figure 10(a) shows G_0 that is the maximal connected 9-truss containing Q. This entire graph has 73 nodes, 486 edges, edge density of 0.18 and diameter of 4. As we can see, most black nodes are at a long distance from each other. They are loosely connected with query nodes via some intermediate nodes. Our method LCTC removes these black nodes and finds a closest truss community for Q shown in Figure 10(b), which is a 9-truss of diameter 2. It has 14 authors, 81 edges and the edge density of 0.89. The community does not include any authors in a 9-truss that are far away from queried authors.

Exp-3 The Quality by Ground-Truth: To evaluate the effectiveness of different community models, we compare LCTC with three other methods MDC, QDC and Truss using the 5 networks, DBLP, Amazon, Youtube, LiveJournal, and Orkut, with ground-truth communities [29]. We randomly select query nodes that appear in a unique ground-truth community, and select 1,000 sets of such query nodes with the size randomly ranging from 1 to 16. We evaluate the accuracy by the F1-score of the detected community, and report the averaged F1-score over all query cases.

Figure 11(a) shows the F1-score. Our method achieves the highest F1-score on most networks. QDC has the second best performance, which outperforms LCTC on Youtube network. MDC does not perform well due to the fixed distance and size constraints. Note that the large overlap of ground-truth communities[29] in Orkut may makes them difficult to be detected accurately by most methods. Figure 11(b) shows that LCTC runs much faster than MDC and QDC, and is close to Truss. Figure 11(c) shows the size of communities detected by LCTC and Truss, in terms of the number of vertices and edges. As we can see, the number of nodes ($|V|$-) and the number of edges ($|E|$-) in communities detected by LCTC are much less than those by Truss on all networks. It confirms the power of eliminating irrelevant nodes from discovered communities by our LCTC.

Exp-4 Diameter and Trussness Approximation: We evaluate the diameter approximation of detected communities by our methods on Facebook network. Here, we take the lower bound of the optimal diameter (LB-OPT) as the smallest query distance $\text{dist}_R(R, Q)$, where R is the community detected by method Basic. We show the curve of $2\text{dist}_R(R, Q)$, which serves as the upper bound of smallest diameter (UB-OPT) by Lemma 2. The averaged diameters of communities detected by different methods are reported in Figure 12(a), where we vary the inter-distance l. The diameters of detected communities obtained by all our methods are very close to the lower bound of optimal one. Figure 12(b) shows the maximum trussness of detected communities by our methods. Basic and BD
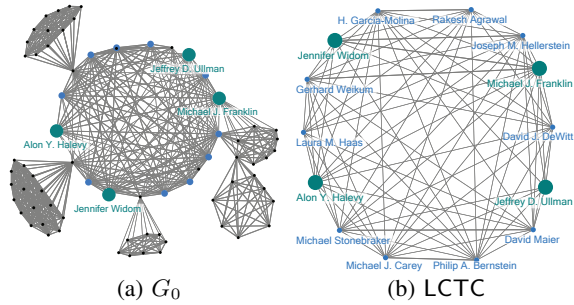
(a) G_0 (b) LCTC

Figure 10: Community search on DBLP network using query $Q =$ {"Alon Y. Halevy", "Michael J. Franklin", "Jeffrey D. Ullman", "Jennifer Widom"}

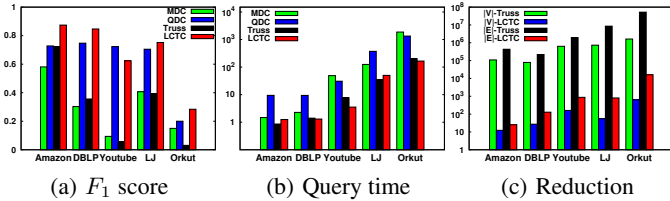(a) F_1 score   (b) Query time   (c) Reduction

Figure 11: Quality evaluation on networks with ground-truth communities

globally search the k-truss containing query nodes on the entire graph, and the detected communities have the maximum trussness k. LCTC can detect the trussness of communities which are very close to Basic and BD, by searching over a small graph locally. Thus, LCTC balances the efficiency and effectiveness well.

7. RELATED WORK AND DISCUSSION

In this section, we firstly discuss the rationale for our proposed model, and then review work most related to our study.

7.1 Design Decisions

Here, we discuss several natural candidates for community models and provide a rationale for our definition of closest truss community.

Diameter vs query distance. Being closely related to the query nodes is a natural desirable property for nodes to be included in a community. However, in the literature, small diameter has been regularly considered as an important hallmark of a good community – see e.g., [11, 10, 14, 12]. Thus, minimizing diameter in identifying communities has a natural motivation.

Secondly, by definition, a community with a small diameter will also have small query distance from its nodes. On the other hand, minimizing query distance ignores the distance between non-query nodes in the community. In this sense, small diameter is a strictly stronger property than small query distance. Example 3 illustrates this point and the value of minimizing diameter as opposed to just query distance.

Trading trussness for diameter. Every k-truss is also a $(k - 1)$-truss by definition. Thus, relaxing the maximum trussness requirement may allow us to find a community with a smaller diameter by sacrificing trussness. One problem is that the variation of diameter as trussness decreases, may not be smooth but may face a sudden drop as trussness decreases to a low value. E.g., continuing with the example of Figure 1(a) with query $Q = \{q_1, q_2, q_3\}$, our CTC model yields a community with the highest trussness $k = 4$ and diameter 3, as in Figure 1(b). When $k = 3$, the 3-truss containing Q with the smallest diameter is still Figure 1(b). However, when $k = 2$, the cycle of $\{(q_1, t), (t, q_3), (q_3, v_4), (v_4, q_2), (q_2, q_1)\}$ turns out to be the 2-truss containing Q and its diameter
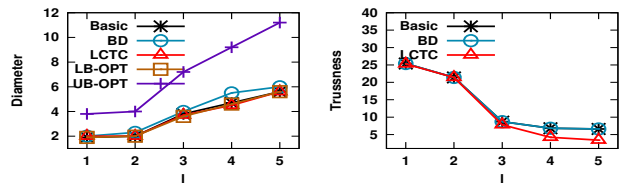

(a) Diameter   (b) Trussness
Figure 12: Varying the inner distance l on Facebook

is 2. However, this is loosely connected and has a low edge density. In general, for a small k, the k-truss community found by removing free riders may have loosely connected structure and thus may be noisy. One advantage of our approach is that it is parameter-free. However, if a user would like to explore trading trussness for diameter, it is straightforward to extend our algorithms (Algorithms 1 and 2) to treat the desired trussness k as a constraint instead of maximizing trussness. Finally, another way of combining trussness and diameter is using a weighted combination, but this comes with the challenge of tuning the weights. Our parameter-free approach of minimizing diameter while keeping trussness at the maximum value is a reasonable choice.

Constraining community size. At first, it appears that we can minimize or avoid free riders by bounding the size of a community. However, sizes of communities may vary widely and it is difficult, if possible at all, to impose proper bounds on acceptable community sizes. Moreover, bounding the size of the community may render the problem of finding a query driven community inapproximable w.r.t. any factor. Specifically, consider the special case of finding a k-truss of size at most a given parameter ℓ that contains $Q = \emptyset$. This subsumes the k-clique problem, which is not approximable within any reasonable factor [15]. By contrast, minimizing diameter instead of size admits efficient approximation. Indeed, our formulation does address community size indirectly. The larger the k, the smaller the size of the k-truss. Our CTC model maximizes trussness. Furthermore, by minimizing the diameter, it helps remove free riders, thus reducing the size in a principled manner. On the algorithmic side, our LCTC method (Section 5.2) actually uses a size threshold to prune the search space and improve efficiency. Thus, LCTC controls the size of a community in a heuristic manner.

7.2 Community Search

Recently, several community search models have been studied, including k-truss [16], quasi-clique [8], k-core [25, 9], influential community [19] and query biased densest subgraph [28]. Here, we compare these models with our proposed closest truss community model w.r.t. three aspects: (i) consideration of query nodes, (ii) cohesive structure, and (iii) quality approximation.

Query nodes. Cui et al. [8] have recently studied the problem of online search of overlapping communities for a query node by designing a new α-adjacency γ-quasi-k-clique model. Huang et al. [16] propose a k-truss community model based on triangle adjacency, to find all overlapping communities of a query node. They ignore the diameter of the resulting community. Cui et al. [9] find a k-core community for a query node using local search. In addition, influential community model [19] finds top-r communities with the highest influence scores over the entire graph; no query nodes are considered. Extending any of above models from one (or zero) query node to multiple query nodes raises new challenges.

Cohesive structure. [25] and [28] support community search of multiple query nodes similarly to us, thus they are most related to our work. Sozio et al. [25] proposed a k-core based community

model, called Cocktail Party model, with distance and size constraints. Our proposed closest truss community model is based on connected k-truss. Conceptually, k-truss is a more cohesive definition than k-core, as k-truss is based on triangles whereas k-core simply considers node degree [27]. Most recently, Wu et al. [28] studied the query biased densest connected subgraph (QDC) problem for avoiding subgraphs irrelevant to query nodes in the community found. While QDC [28] is also defined based on a connected graph containing Q similarly to CTC, it optimizes a fundamentally different function called query biased edge density, which is calculated as the overall edge weight averaged over the weight of nodes in a community.

Quality approximation. Both problems proposed in [25] and [28] are NP-hard to compute, and do not admit approximations without further assumptions. [28] gives an approximation solution of QDC by relaxing the problem. Unfortunately, as the authors show themselves [28], this could fail in real applications, for two reasons. First, the algorithm may find a solution consisting of several connected components with query nodes split between them. Second, the approximation factor can be large, which can deteriorate further with a larger number of query nodes. In contrast, we provide an efficient 2-approximation algorithm for finding the closest truss community containing any set of query nodes.

7.3 Community Detection

The goal of community detection is to identify all communities in the entire network. A typical method for finding communities is to optimize the modularity measure [22]. Generally, community detection falls into two major categories: non-overlapping [24, 30] and overlapping community detection [23, 1]. All these methods consider static communities, where the networks are partitioned a priori. Query nodes are not considered since their focus is not community search. Thus these works on community detection are significantly different from our goal of query driven community search.

7.4 Dense Subgraph Mining

There is a very large body of work on mining dense subgraph patterns, including clique [3], quasi-clique [26], k-core [2, 4], k-truss [6, 27] to name a few. Various studies have been done on core decomposition and truss decomposition in different settings, including in-memory algorithms [2, 6], external-memory algorithms [4, 27], and MapReduce [7]. None of these works considers query nodes, which as we have discussed earlier, raise major computational challenges.

ACKNOWLEDGMENTS

8. CONCLUSION

In this paper, we study the closest truss community search problem over a graph, i.e., given a set of query nodes, we find a densely connected community containing the query nodes, in which nodes are close to each other. Based on the dense subgraph definition as a k-truss, we formulate the CTC as a connected k-truss subgraph containing the query nodes with the largest k, and having the minimum diameter among such subgraphs. We developed a greedy

algorithmic framework that provides a 2-approximation to the optimal solution, and also designed several efficient search strategies, such as using a truss index and bulk deletion optimization and local exploration. Extensive experimental results on large real-world networks with ground-truth communities demonstrate the effectiveness and efficiency of our proposed solutions.

9. REFERENCES

[1] Y.-Y. Ahn, J. P. Bagrow, and S. Lehmann. Link communities reveal multiscale complexity in networks. *Nature*, 466(7307):761–764, 2010.

[2] V. Batagelj and M. Zaversnik. An o (m) algorithm for cores decomposition of networks. *arXiv preprint cs/0310049*, 2003.

[3] C. Bron and J. Kerbosch. Finding all cliques of an undirected graph (algorithm 457). *Commun. ACM*, 16(9):575–576, 1973.

[4] J. Cheng, Y. Ke, S. Chu, and M. T. Özsu. Efficient core decomposition in massive networks. In *ICDE*, pages 51–62, 2011.

[5] N. Chiba and T. Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14(1):210–223, 1985.

[6] J. Cohen. Trusses: Cohesive subgraphs for social network analysis. Technical report, National Security Agency, 2008.

[7] J. Cohen. Graph twiddling in a mapreduce world. *Computing in Science and Engineering*, 11(4):29–41, 2009.

[8] W. Cui, Y. Xiao, H. Wang, Y. Lu, and W. Wang. Online search of overlapping communities. In *SIGMOD*, pages 277–288, 2013.

[9] W. Cui, Y. Xiao, H. Wang, and W. Wang. Local search of communities in large graphs. In *SIGMOD*, pages 991–1002, 2014.

[10] S. R. Doddi, M. V. Marathe, S. Ravi, D. S. Taylor, and P. Widmayer. Approximation algorithms for clustering to minimize the sum of diameters. In *Algorithm Theory-SWAT 2000*, pages 237–250. Springer, 2000.

[11] J. Edachery, A. Sen, and F. J. Brandenburg. Graph clustering using distance-k cliques. In *Proceedings of the 7th International Symposium on Graph Drawing*, pages 98–106, 1999.

[12] J. Edachery, A. Sen, and F. J. Brandenburg. Graph clustering using distance-k cliques. In *Graph drawing*, pages 98–106. Springer, 1999.

[13] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[14] T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.

[15] J. Håstad. Clique is hard to approximate within n 1-&epsiv. In *Foundations of Computer Science, 1996. Proceedings., 37th Annual Symposium on*, pages 627–636. IEEE, 1996.

[16] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu. Querying k-truss community in large and dynamic graphs. In *SIGMOD*, pages 1311–1322, 2014.

[17] X. Huang, L. V. Lakshmanan, J. X. Yu, and H. Cheng. Approximate closest community search in networks. *arXiv:1505.05956*, 2015.

[18] L. Kou, G. Markowsky, and L. Berman. A fast algorithm for steiner trees. *Acta informatica*, 15(2):141–145, 1981.

[19] R.-H. Li, L. Qin, J. X. Yu, and R. Mao. Influential community search in large networks. *PVLDB*, 8(5), 2015.

[20] J. J. McAuley and J. Leskovec. Learning to discover social circles in ego networks. In *NIPS*, volume 272, pages 548–556, 2012.

[21] K. Mehlhorn. A faster approximation algorithm for the steiner problem in graphs. *Information Processing Letters*, 27(3):125–128, 1988.

[22] M. E. Newman. Fast algorithm for detecting community structure in networks. *Physical review E*, 69(6):066133, 2004.

[23] G. Palla, I. Derényi, I. Farkas, and T. Vicsek. Uncovering the overlapping community structure of complex networks in nature and society. *Nature*, 435(7043):814–818, 2005.

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

[25] M. Sozio and A. Gionis. The community-search problem and how to plan a successful cocktail party. In *KDD*, pages 939–948, 2010.

[26] C. E. Tsourakakis, F. Bonchi, A. Gionis, F. Gullo, and M. A. Tsiarli. Denser than the densest subgraph: Extracting optimal quasi-cliques with quality guarantees. In *KDD*, pages 104–112, 2013.

[27] J. Wang and J. Cheng. Truss decomposition in massive networks. *PVLDB*, 5(9):812–823, 2012.

[28] Y. Wu, R. Jin, J. Li, and X. Zhang. Robust local community detection: On free rider effect and its elimination. *PVLDB*, 8(7), 2015.

[29] J. Yang and J. Leskovec. Defining and evaluating network communities based on ground-truth. In *ICDM*, pages 745–754, 2012.

[30] Z. Yang, T. Hao, O. Dikmen, X. Chen, and E. Oja. Clustering by nonnegative matrix factorization using graph random walk. In *NIPS*, pages 1079–1087, 2012.