Attribute-Driven Community Search

Xin Huang*
Hong Kong Baptist University
Hong Kong, China
xinhuang@comp.hkbu.edu.hk

Laks V.S. Lakshmanan
University of British Columbia
Vancouver, Canada
laks@cs.ubc.ca

ABSTRACT

Recently, community search over graphs has gained significant interest. In applications such as analysis of protein-protein interaction (PPI) networks, citation graphs, and collaboration networks, nodes tend to have attributes. Unfortunately, most previous community search algorithms ignore attributes and result in communities with poor cohesion w.r.t. their node attributes. In this paper, we study the problem of attribute-driven community search, that is, given an undirected graph G where nodes are associated with attributes, and an input query Q consisting of nodes V_q and attributes W_q, find the communities containing V_q, in which most community members are densely inter-connected and have similar attributes.

We formulate this problem as finding attributed truss communities (ATC), i.e., finding connected and close k-truss subgraphs containing V_q, with the largest attribute relevance score. We design a framework of desirable properties that good score function should satisfy. We show that the problem is NP-hard. However, we develop an efficient greedy algorithmic framework to iteratively remove nodes with the least popular attributes, and shrink the graph into an ATC. In addition, we also build an elegant index to maintain k-truss structure and attribute information, and propose efficient query processing algorithms. Extensive experiments on large real-world networks with ground-truth communities show that our algorithms significantly outperform the state of the art and demonstrates their efficiency and effectiveness.

1. INTRODUCTION

Graphs have emerged as a powerful model for representing different types of data, such as protein-protein interaction networks, sensor/communication networks, and collaboration networks. In these graphs, communities naturally exist as groups of vertices that are densely interconnected. Consequently, community detection, i.e., finding all communities in a given network, serves as a global network-wide analysis tool, and has been extensively studied in the literature. More recently, a related but different problem called community search has generated considerable interest. It is motivated by the need to make answers more meaningful and personalized to the user [27, 16]. For a given set of query nodes, community search seeks to find the communities containing the query nodes.

*This work was partially done by the author while he was at the University of British Columbia.

Proceedings of the VLDB Endowment, Vol. 10, No. 9

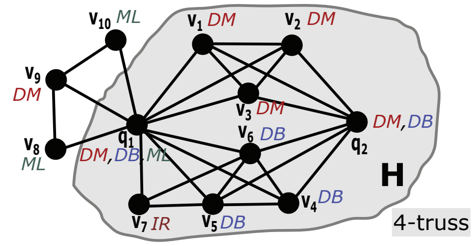
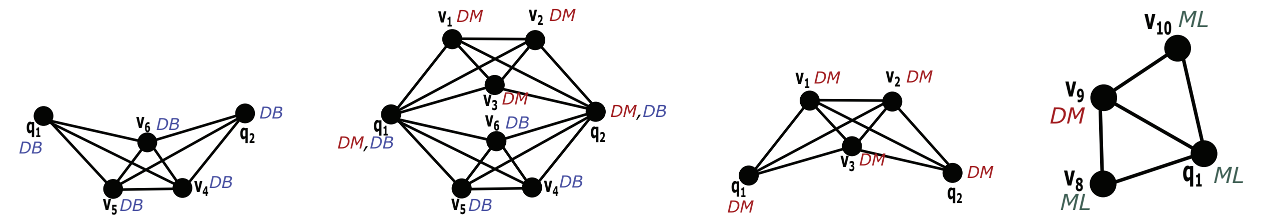


Figure 1: An example attributed graph G

In the aforementioned applications, the entities modeled by the network nodes often have properties which are important for making sense of communities. E.g., authors in collaboration networks have areas of expertise; proteins have molecular functions, biological processes, and cellular components as properties. Such networks can be modeled using *attributed graphs* [37] where attributes associated with nodes capture their properties. E.g., Figure 1 shows an example of a collaboration network. The nodes $q_i, v_j, ...$ represent authors. Node attributes (e.g., DB, ML) represent authors' topics of expertise. In finding communities (with or without query nodes) over attributed graphs, we might want to ensure that the nodes in the discovered communities have homogeneous attributes. For instance, it has been found that communities with homogeneous attributes among nodes more accurately predict protein complexes [15]. Furthermore, we might wish to query, not just using query nodes, but also using query attributes. To illustrate, consider searching for communities containing the nodes $\{q_1, q_2\}$. Based on structure alone, the subgraph H shown in Figure 1 is a good candidate answer for this search, as it is densely connected. However, attributes of the authors in this community are not homogeneous: the community is a mix of authors working in different topics – DB, DM, IR, and ML. Previous community search methods include those based on k-core [32, 26, 10], k-truss [19], and 1.0-quasi-k-clique-ℓ-adjacent community [9]. A k-core [26] is a subgraph in which each vertex has at least k neighbors within the subgraph. A k-truss [19] is a subgraph in which each edge is contained in at least $(k-2)$ triangles within the subgraph. The 1.0-quasi-k-clique-ℓ-adjacent community model [9] allows two k-cliques overlapping in ℓ vertices to be merged into one community. In Figure 1, for $k = 4$ and $\ell = 3$, all these community models will report H as the top answer and are thus unsatisfactory. Thus, in general, communities found by most previous community search methods can be hard to interpret owing to the heterogeneity of node attributes. Furthermore, the communities reported could contain smaller dense subgraphs with more homogeneity in attributes, which are missed by most previous methods. A recent work [13] proposed an at-



(a) H_1. 4-truss community on V_q={q_1, q_2}, W_q={DB} (b) H_2. 4-truss community on V_q={q_1,q_2}, W_q={DB, DM} (c) H_3. 4-truss community on V_q={q_1, q_2}, W_q={DM} (d) H_4. 3-truss community on V_q={q_1}, W_q={ML}

Figure 2: Attributed Communities for queries on different query nodes V_q and query attributes W_q.

tributed community model. A detailed comparison of [13] with our model appears in Section 3.

Consider now querying the graph of Figure 1 with query nodes $\{q_1, q_2\}$ *and* attributes (i.e., keywords) {DB, DM}. We would expect this search to return subgraph H_2 (Figure 2(b)). On the other hand, for the same query nodes, if we search with attribute {DB} (resp., {DM}), we expect the subgraph H_1 (resp., H_3) to be returned as the answer (Figure 2(a)&(c)). Both H_1 and H_3 are dense subgraphs where all authors share a common topic (DB or DM).

Given a query consisting of nodes and attributes (keywords), one may wonder whether we can filter out nodes not having those attributes and then run a conventional community search method on the filtered graph. To see how well this may work, consider querying the graph in Figure 1 with query node q_1 and query attribute ML. Filtering out nodes without attribute ML and applying community search yields the chain consisting of v_{10}, q_1, v_8, which is not densely connected. On the other hand, the subgraph induced by $\{q_1, v_8, v_9, v_{10}\}$ is a 3-truss in Figure 2(d). Even though it includes one node without ML it is more densely connected than the chain above and is a better answer than the chain as it brings out denser collaboration structure among the authors in the community. Thus, a simple filtering based approach will not work.

In sum, attributed graphs present novel opportunities for community search by combining dense structure of subgraphs with the level of homogeneity of node attributes in the subgraph. Most previous work in community search fails to produce satisfactory answers over attributed graphs, while keyword search based techniques do not find dense subgraphs. This raises the following major challenges for community search in attributed graphs. Firstly, how should we combine dense connectedness with the distribution of attributes over the community nodes? We need a community definition that promotes dense structure as well as attribute homogeneity. However, there can be tension between these goals: as illustrated in the example above, some denser subgraphs may be less homogeneous in their node attributes than some sparser ones. Secondly, the definition should capture the intuition that the more input attributes that are covered by a community, the better the community. Finally, the query processing algorithm should be efficient for large graphs.

To tackle these challenges, we propose an attributed truss community (ATC) model. Given a query $Q = (V_q, W_q)$ consisting of a set of query nodes V_q and a set of query attributes W_q, a good community H must be a dense subgraph which contains all query nodes and attributes W_q must be contained in numerous nodes of the community. The more nodes with attribute $w \in W_q$, the more importance to w commonly accorded by the community members. Additionally, the nodes must share as many attributes as possible. Notice that these two conditions are not necessarily equivalent. Capturing these intuitions, we define an attribute score function that strikes a balance between attribute homogeneity and coverage. Moreover, as a qualifying cohesive and tight structure, we define a novel concept of (k, d)-truss for modeling a densely connected community. A (k, d)-truss is a connected k-truss containing all query nodes, where each node has a distance no more than d from every query node. This inherits many nice structural properties, such as bounded diameter, k-edge connectivity, and hierarchical structure. Thus, based on attribute score function and (k, d)-truss, we propose a novel community model as **attributed truss community** (ATC), which is a (k, d)-truss with the maximum attribute score. In this paper, we make the following contributions.

- We motivate the problem of attributed community search, and identify the desiderata of a good attributed community (Section 2).
- We propose a novel dense and tight subgraph, (k, d)-truss, and design an attribute score function satisfying the desiderata set out above. Based on this, we propose ATC community model, and formulate the problem (Section 4).
- We analyze the structural properties of ATC and show that it is non-monotone, non-submodular and non-supermodular, which signal huge computational challenges. We also formally prove that the problem is NP-hard (Section 5).
- We develop a greedy algorithmic framework to find an ATC containing given query nodes w.r.t. given query attributes. It first finds a maximal (k, d)-truss, and then iteratively removes nodes with smallest attribute score contribution. For improving the efficiency and quality, we design a revised attribute marginal gain function and a bulk removal strategy for cutting down the number of iterations (Section 6).
- For further improving efficiency, we explore the local neighborhood of query nodes to search an ATC. This algorithm first generates a Steiner tree connecting all query nodes, and then expands the tree to a dense subgraph with the insertion of carefully selected nodes, that have highly correlated attributes and densely connected structure (Section 7).
- We conduct extensive experiments on 7 real datasets, and show that our attributed community model can efficiently and effectively find ground-truth communities and social circles over real-world networks, significantly outperforming previous work (Section 8).

We discuss related work in Section 3, and conclude the paper with a summary in Section 9. For lack of space, some proofs and additional experiments are omitted. The complete details can be found in the full version of the paper [18].

2. PRELIMINARIES AND DESIDERATA

2.1 Preliminaries

We consider an undirected, unweighted simple graph $G = (V, E)$ with $n = |V(G)|$ vertices and $m = |E(G)|$ edges. We denote the set of neighbors of a vertex v by $N(v)$, and the degree of v by $d(v) = |N(v)|$. We let $d_{max} = \max_{v \in V} d(v)$ denote the maximum vertex degree in G. W.l.o.g. we assume that the graphs we consider are connected. Note that this implies that $m \geq n - 1$. We consider attributed graphs and denote the set of all attributes in a graph by $\mathcal{A}$. Each node $v \in V$ contains a set of zero or more attributes, denoted by $\mathsf{attr}(v) \subseteq \mathcal{A}$. The multiset union of attributes of all nodes in G is denoted $\mathsf{attr}(V)$. Note that $|\mathsf{attr}(V)| = \sum_{v \in V} |\mathsf{attr}(v)|$. We use $V_w \subseteq V$ to denote the set of nodes having attribute w, i.e., $V_w = \{v \in V \mid w \in \mathsf{attr}(v)\}$.

2.2 Desiderata of a good community

Given a query $Q = (V_q, W_q)$ with a set of query nodes $V_q \subseteq V$ and a set of query attributes W_q, the attributed community search (ACS) problem is to find a subgraph $H \subseteq G$ containing all query nodes V_q, where the vertices are densely inter-connected, cover as many query attributes W_q as possible and share numerous attributes. In addition, the communication cost of H should be low. We call the query $Q = (V_q, W_q)$ an ACS query. Before formalizing the problem, we first identify the commonly accepted desiderata of a good attributed community.

Criteria of a good attributed community: Given a graph $G(V, E)$ and a ACS query $Q = (V_q, W_q)$, an attributed community is a connected subgraph $H = (V(H), E(H)) \subseteq G$ that satisfies:

1. (Participation) H contains all query nodes as $V_q \subseteq V(H)$;
2. (Cohesiveness) A cohesiveness function $\mathsf{coh}(H)$ that measures the cohesive structure of H is high.
3. (Attribute Coverage and Correlation) An attribute score function $\mathsf{f}(H, W_q)$ that measures the coverage and correlation of query attributes in vertices of H is high.
4. (Communication Cost) A communication cost function $\mathsf{com}(H)$ that measures the distance of vertices in H is low.

The participation condition is straightforward. The cohesiveness condition is also straightforward since communities are supposed to be densely connected subgraphs. One can use any notion of dense subgraph previously studied, such as k-core, k-truss, etc. The third condition captures the intuition that more query attributes covered by H, the higher $\mathsf{f}(H, W_q)$; also more attributes shared by vertices of H, the higher $\mathsf{f}(H, W_q)$. This motivates designing functions $\mathsf{f}(.,.)$ with this property. Finally, keeping the communication cost low helps avoid irrelevant vertices in a community. This is related to the so-called free rider effect, studied in [19, 34]. Intuitively, the closer the community nodes to query nodes, subject to all other conditions, the more relevant they are likely to be to the query. Notice that sometimes a node that does not contain query attributes may still act as a "bridge" between other nodes and help improve the density. A general remark is that other than the first condition, for conditions 2–4, we may either optimize a suitable metric or constrain that the metric be above a threshold (below a threshold for Condition 4). We formalize this intuition in Section 4 and give a precise definition of an attributed community .

3. RELATED WORK

Work related to this paper can be classified into community search, keyword search, team formation, and community detection in attributed graphs. Table 1 shows a detailed comparison of representative works on these topics.

Community Search. Community search on a graph aims to find densely connected communities containing query nodes, and has attracted a great deal of attention recently. Various models based on different dense subgraphs [32, 16, 31] have been proposed and studied: quasi-clique [9], densest subgraph [34], k-core [32, 10, 3] and k-truss [16, 19]. All these works focus on the structure of the community while ignoring node attributes. This can result in communities with poor cohesion in the attribute sets of the community nodes. In particular, while [16, 19] use k-truss as the basis structure of communities, the k-truss communities they find are not guaranteed to have high cohesion in the attribute sets of the nodes.

Table 1: A comparison of representative works on keyword search (KS), team formation (TF), community search (CS) and attributed community search (ACS).

Method	Topic	Participation Condition	Attribute Function	Cohesiveness Constraint	Communication Cost
[4]	KS	χ	✓	χ	✓
[25]	KS	χ	✓	χ	✓
[24]	TF	χ	✓	χ	✓
[21]	TF	χ	✓	χ	✓
[32]	CS	✓	χ	✓	✓
[19]	CS	✓	χ	✓	✓
[13]	ACS	✓	✓	✓	χ
Ours	ACS	✓	✓	✓	✓

Keyword Search. Keyword search in relational databases has been extensively studied. Most of the works focus on finding minimal connected tuple trees from a relational database [1, 14, 20, 11]. Keyword search over graphs finds a substructure containing all or a subset of the input keywords. The works [25, 29] report subgraphs instead of trees as keyword search answers. However, keyword search does not consider the cohesive structure involving the query nodes and keywords.

Team Formation. Lappas et al. [24] introduced the problem of discovering a team of experts from a social network, that satisfies all attributed skills required for a given task with low communication cost. Most of the team formation studies [21] focus on a tree substructure, as opposed to densely connected subgraph required by community search. Compared with our problem, these studies do not consider both dense structure and distance constraint at the same time, and also have no constraint on query nodes.

Community Detection in Attributed Graphs. Community detection in attributed graphs is to find all densely connected components with homogeneous attributes [37, 6, 30, 17]. A survey of clustering on attributed graphs can be found in [5]. It is practically hard and inefficient to adapt the above community detection approaches [37, 36, 17, 30] for online attributed community search: community detection is inherently global and much of the work involved may be irrelevant to the community being searched.

Recently, Yang et al. [13] have proposed a model for community search over attributed graphs based on k-cores. The key distinction with our work is as follows. (1) Our community model is based on k-trusses, which have well-known advantages over k-cores such as denser structure. A connected k-core has no guarantee to be 2-edge-connected, even with a large core value k. (2) Our search supports multiple query nodes whereas theirs is limited to a single query node. (3) Their approach may miss useful communities. E.g., consider the example graph in Figure 1 with query node $\{q_2\}$ *and* attributes {DB, DM}, and parameter $k = 3$. Their model will return the subgraphs H_1 (Figure 2(a)) and H_3 (Figure 2(c)) as answers. However, the subgraph H_2 (Figure 2(a)) will not be discovered, due to their strict homogeneity constraints. (4) Finally, unlike them, we validate our model with experiments over datasets with ground-truth communities.

4. ATTRIBUTED COMMUNITY MODEL

In this section, we develop a notion of attributed community by formalizing the the desiderata discussed in Section 2. We focus our discussion on conditions 2–4.

4.1 (k, d)-truss

In the following, we introduce a novel definition of dense and tight substructure called (k, d)-truss by paying attention to cohesiveness and communication cost.

Cohesiveness. While a number of definitions for dense subgraphs have been proposed over the years, we adopt the k-truss model, proposed by Cohen [8], which has gained popularity and has been found to satisfy nice properties.

A subgraph $H \subseteq G$ is a k-core, if every vertex in H has degree at least k. A *triangle* in G is a cycle of length 3. We denote a triangle involving vertices $u, v, w \in V$ as $\triangle_{uvw}$. The *support* of an edge $e(u, v) \in E$ in G, denoted $sup_G(e)$, is the number of triangles containing e, i.e., $sup_G(e) = |\{\triangle_{uvw} : w \in V\}|$. When the context is obvious, we drop the subscript and denote the support as $sup(e)$. Since the definition of k-truss [8, 33] allows a k-truss to be disconnected, we define a connected k-truss below.

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 in which each connection (edge) (u, v) is "endorsed" by $k-2$ common neighbors of u and v [8]. A connected k-truss with a large value of k signifies strong inner-connections between members of the subgraph. In a k-truss, each node has degree at least $k-1$, i.e., it is a $(k-1)$-core.

Consider the graph G (Figure 1). The edge $e(v_1, v_2)$ is contained in three triangles $\triangle_{q_1v_1v_2}$, $\triangle_{q_2v_1v_2}$ and $\triangle_{v_3v_1v_2}$, thus its support is $sup_G(e) = 3$. Consider the subgraph H_3 of G (Figure 2(c)). Every edge of H_3 has support ≥ 2, thus H_3 is a 4-truss. Note that even though the edge $e(v_1, v_2)$ has support 3, there exists no 5-truss in the graph G in Figure 1.

Communication Cost. For two nodes $u, v \in G$, let $\mathsf{dist}_G(u, v)$ denote the length of the shortest path between u and v in G, where $\mathsf{dist}_G(u, v) = +\infty$ if u and v are not connected. The diameter of a graph G is the maximum length of a shortest path in G, i.e., $\mathsf{diam}(G) = \max_{u,v \in G}\{\mathsf{dist}_G(u, v)\}$. We make use of the notion of graph query distance in the following.

DEFINITION 2 (QUERY DISTANCE [19]). *Given a graph G and query nodes $V_q \subseteq V$, the vertex query distance of vertex $v \in V$ is the maximum length of a shortest path from v to a query node $q \in V_q$ in G, i.e., $\mathsf{dist}_G(v, V_q) = \max_{q \in V_q} \mathsf{dist}_G(v, q)$. Given a subgraph $H \subseteq G$ and $V_q \subseteq V(H)$, the graph query distance of H is defined as $\mathsf{dist}_H(H, V_q) = \max_{u \in H} \mathsf{dist}_H(u, V_q)$ $= \max_{u \in H, q \in V_q} \mathsf{dist}_H(u, q)$.*

Given a subgraph $H \subseteq G$, the query distance $\mathsf{dist}_H(H, V_q)$ measures the communication cost between the members of H and the query nodes. A good community should have a low communication cost with small $\mathsf{dist}_H(H, V_q)$.

For the graph G in Figure 1 and query nodes $V_q = \{q_1, q_2\}$, the vertex query distance of v_7 is $\mathsf{dist}_G\ (v_7, V_q) = \max_{q \in V_q} \{\mathsf{dist}_G(v_7, q)\} = 2$. Consider the subgraph H_1 in Figure 2(a). Then graph query distance of H_1 is $\mathsf{dist}_{H_1}(H_1, V_q) = \mathsf{dist}_{H_1}\ (q_1, q_2) = 2$. The diameter of H_1 is $\mathsf{diam}(H_1) = 2$.

(k, d)-truss. We adapt the notions of k-truss and query distance, and propose a new notion of (k, d)-truss capturing dense cohesiveness and low communication cost.

DEFINITION 3 ((k, d)-truss). *Given a graph H, query nodes V_q, and numbers k and d, we say that H is a (k, d)-truss iff H is a connected k-truss containing V_q and $\mathsf{dist}_H(H, V_q) \leq d$.*

By definition, the cohesiveness of a (k, d)-truss increases with k, and its proximity to query nodes increases with decreasing d. For instance, the community H_1 in Figure 2 (a) for $V_q = \{q_1, q_2\}$ is a (k, d)-truss with $k = 4$ and $d = 2$.

4.2 Attribute Score Function

We first identify key properties that should be obeyed by a good attribute score function for a community. Let $\mathsf{f}(H, W_q)$ denote the attribute score of community H w.r.t. query attributes W_q. We say that a node v of H covers an attribute $w \in W_q$, if $w \in \mathsf{attr}(v)$. We say that a node of H is irrelevant to the query if it does not cover any of the query attributes.

Principle 1: The more query attributes that are covered by some node(s) of H, the higher should be the score $\mathsf{f}(H, W_q)$. The rationale is obvious.

Principle 2: The more nodes contain an attribute $w \in W_q$, the higher the contribution of w should be toward the overall score $\mathsf{f}(H, W_q)$. The intuition is that attributes that are covered by more nodes of H signify homogeneity within the community w.r.t. shared query attributes.

Principle 3: The more nodes of H that are irrelevant to the query, the lower the score $\mathsf{f}(H, W_q)$.

We next discuss a few choices for defining $\mathsf{f}(H, W_q)$ and analyze their pros and cons, before presenting an example function that satisfies all three principles. Note that the scores $\mathsf{f}(H, W_q)$ are always compared between subgraphs H that meet the same structural constraint of (k, d)-truss. An obvious choice is to define $\mathsf{f}(H, W_q) := \sum_{w \in W_q} \mathsf{score}(H, w)$, where $\mathsf{score}(H, w)$, the contribution of attribute w to the overall score, can be viewed as the relevance of H w.r.t. w. This embodies Principle 1 above. Inspired by Principle 2, we could define $\mathsf{score}(H, w) := |V(H) \cap V_w|$, i.e., the number of nodes of H that cover w. Unfortunately, this choice suffers from some limitations by virtue of treating all query attributes alike. Some attributes may not be shared by many community nodes while others are and this distinction is ignored by the above definition of $\mathsf{f}(H, W_q)$. To illustrate, consider the community H_1 in Figure 2(a) and the query $Q = (\{q_1\}, \{DB\})$; H_1 has 5 vertices associated with the attribute DB and achieves a score of 5. The subgraph H of the graph G shown in Figure 1 also has the same score of 5. However, while the community in Figure 2(a) is clearly a good community, as all nodes carry attribute DB, the subgraph H in Figure 1 includes several irrelevant nodes without attribute DB. Notice that both H_1 and H are 4-trusses so we have no way of discriminating between them, which is undesirable.

An alternative is to define $\mathsf{score}(H, w)$ as $\frac{|V_w \cap V(H)|}{|V(H)|}$ as this captures the popularity of attribute w. Unfortunately, this fails to reward larger commmunities. For instance, consider the query $Q = (\{q_1, v_4\}, \{DB\})$ over the graph G in Figure 1. The subgraph H_1 in Figure 2(a) as well as its subgraph obtained by removing q_2 is a 4-truss and both will be assigned a score of 1.

In view of these considerations, we define $\mathsf{f}(H, W_q)$ as a weighted sum of the score contribution of each query attribute, where the weight reflects the popularity of the attribute.

DEFINITION 4 (ATTRIBUTE SCORE). *Given a subgraph $H \subseteq G$ and an attribute w, the weight of an attribute w is $\theta(H, w) = \frac{|V_w \cap V(H)|}{|V(H)|}$, i.e., the fraction of nodes of H covering w. For a query $Q = (V_q, W_q)$ and a community H, the attribute score of H is defined as $\mathsf{f}(H, W_q) = \sum_{w \in W_q} \theta(H, w) \times \mathsf{score}(H, w)$, where $\mathsf{score}(H, w) = |V_w \cap V(H)|$ is the number of nodes covering w.*

The contribution of an attribute w to the overall score is $\theta(H, w) \times \mathsf{score}(H, w) = \frac{|V_w \cap V(H)|^2}{|V(H)|}$. This depends not only on the number of vertices covering w but also on w's popularity in the community H. This choice discourages vertices unrelated to the query attributes W_q which decrease the relevance score, without necessarily increasing the cohesion (e.g., trussness). At the same time, it permits the inclusion of essential nodes, which are added to a community to reduce the cost of connecting query nodes. They act as an important link between nodes that are related to the query, leading to a higher relevance score. We refer to such additional nodes as *steiner nodes*. E.g., consider the query $Q = (\{q_1\}, \{ML\})$ on the graph G in Figure 1. As discussed in Section 1, the community H_4 in Figure 2(d) is preferable to the chain of nodes v_8, q_1, v_{10}. Notice that it includes v_9 with attribute DM (but not ML); v_9 is thus a steiner node. It can be verified that $\mathsf{f}(H_4, W_q) = \frac{9}{4}$ which is smaller than the attribute score of the chain, which is 3. However, H_4 is a 3-truss whereas the chain is a 2-truss. It is easy to see that any supergraph of H_4 in Figure 1 is at most a 3-truss and has a strictly smaller attribute score.

The more query attributes a community has that are shared by more of its nodes, the higher its attribute score. For example, consider the query $Q = (\{q_1\}, \{DB, DM\})$ on graph of Figure 1. The communities H_1, H_2, H_3 in Figure 2 are all potential answers for this query. We find that $\mathsf{f}(H_1, W_q) = 5 \cdot 1 + 2 \cdot \frac{2}{5} = 5.8$; by symmetry, $\mathsf{f}(H_3, W_q) = 5.8$; on the other hand, $\mathsf{f}(H_2, W_q) = 5 \cdot \frac{5}{8} + 5 \cdot \frac{5}{8} = 6.25$. Intuitively, we can see that H_1 and H_3 are mainly focused in one area (DB or DM) whereas H_2 has 5 nodes covering DB and DM each and also has the highest attribute score.

REMARK 1. *We stress that the main contribution of this subsection is the identification of key principles that an attribute score function must satisfy in order to be effective in measuring the goodness of an attributed community. Specifically, these principles capture the important properties of high attribute coverage and high attribute correlation within a community and minimal number of nodes irrelevant to given query. Any score function can be employed as long as it satisfies these principles. The algorithmic framework we propose in Section 6.1 is flexible enough to handle an* ATC *community model equipped with any such score function.*

We note that a natural candidate for attribute scoring is the entropy-based score function, defined as $\mathsf{f}_{\mathsf{entropy}}(H, W_q) = \sum_{w \in W_q} -\frac{|V_w \cap V(H)|}{|V(H)|} \log \frac{|V_w \cap V(H)|}{|V(H)|}$. *It measures homogeneity of query attributes very well. However, it fails to reward larger communities, specifically violating Principle 1. E.g., consider the query* $Q = (\{q_1, v_4\}, \{DB\})$ *on the graph G in Figure 1. The subgraph* H_1 *in Figure 2(a) and its subgraph obtained by removing* q_2 *are both 4-trusses and both are assigned a score of 0. Clearly,* H_1 *has more nodes containing the query attribute DB.*

4.3 Attributed Truss Community Model

Combining the structure constraint of (k, d)-truss and the attribute score function $\mathsf{f}(H, W_q)$, we define an *attributed truss community* (ATC) as follows.

DEFINITION 5. *[Attribute Truss Community] Given a graph G and a query* $Q = (V_q, W_q)$ *and two numbers k and d, H is an attribute truss community (*ATC*), if H satisfies the following conditions:*

1. *H is a* (k, d)-truss *containing* V_q.
2. *H has the maximum attribute score* $\mathsf{f}(H, W_q)$ *among subgraphs satisfying condition (1).*

In terms of structure and communication cost, condition (1) not only requires that the community containing the query nodes V_q be densely connected, but also that each node be close to the query nodes. In terms of query attribute coverage and correlation, condition (2) ensures that as many query attributes as possible are covered by as many nodes as possible.

EXAMPLE 1. *For the graph G in Figure 1, and query* $Q = (\{q_1, q_2\}, \{DB, DM\})$ *with* $k = 4$ *and* $d = 2$, H_2 *in 2(b) is the corresponding* ATC*, since* H_2 *is a* $(4, 2)$*-truss with the largest score* $\mathsf{f}(H, W_q) = 6.25$ *as seen before.*

The ATC-Problem studied in this paper can be formally formulated as follows.

Problem Statement: Given a graph $G(V, E)$, query $Q = (V_q, W_q)$ and two parameters k and d, find an ATC H, such that H is a (k, d)-truss with the maximum attribute score $\mathsf{f}(H, W_q)$.

5. PROBLEM ANALYSIS

In this section, we analyze the complexity of the problem and show that it is NP-hard. We then analyze the properties of the structure and attribute score function of our problem. Our algorithms for community search exploit these properties.

5.1 Hardness

Our main result in this section is that the ATC-Problem is NP-hard (Theorem 2). The crux of our proof idea comes from the hardness of finding the densest subgraph with $\geq k$ vertices [22]. Unfortunately, that problem cannot be directly reduced to our ATC-Problem. To bridge this gap, we extend the notion of graph density to account for vertex weights and define a helper problem called WDalK-Problem – given a graph, find the subgraph with maximum "weighed density" with at least k vertices.

Weighted Density. Let $G = (V, E)$ be an undirected graph. Let $\mathsf{w}(v)$ be a non-negative weight associated with each vertex $v \in V$. Given a subset $S \subseteq V$, the subgraph of G induced by S is $G_S = (S, E(S))$, where $E(S) = \{(u, v) \in E \mid u, v \in S\}$. For a vertex v in a subgraph $H \subseteq G$, its degree is $\deg_H(v) = |\{(u, v) \mid (u, v) \in E(H)\}|$. Next, we define:

DEFINITION 6 (WEIGHTED DENSITY.). *Given a subset of vertices* $S \subseteq V$ *of a weighted graph G, the weighted density of subgraph* G_S *is defined as* $\chi(G_S) = \sum_{v \in S} \frac{\deg_{G_S}(v) + \mathsf{w}(v)}{|S|}$.

Recall that traditional edge density of an induced subgraph G_S is $\rho(G_S) = \frac{|E(S)|}{|S|} = \sum_{v \in S} \frac{\deg_{G_S}(v)}{2|S|}$ [22, 2]. That is, $\rho(G_S)$ is twice the average degree of a vertex in G_S. Notice that in Definition 6, if the weight of v is $\mathsf{w}(v) = 0$, $\forall v$, then the weighted density $\chi(G_S) = 2\rho(G_S)$. It is well known that given a number k, finding the maximum density of a subgraph G_S containing at least k vertices is NP-hard [22].

Define the weight of a vertex v in a graph G as its degree in G, i.e., $\mathsf{w}(v) = \deg_G(v)$. Then, $\chi(G_S) = \sum_{v \in S} \frac{\deg_{G_S}(v) + \deg_G(v)}{|S|} = 2\rho(G_S) + \sum_{v \in S} \frac{\deg_G(v)}{|S|}$. We define a problem, the WDalK-Problem, as follows: given a graph G with weights as defined above, and a density threshold α, check whether G contains an induced subgraph H with at least k vertices such that $\chi(H) \geq \alpha$.

THEOREM 1. WDalK-Problem *is NP-hard.*

PROOF. We reduce the well-known NP-hard problem of Maximum Clique (decision version) to this problem. A complete proof is reported in the arXiv article [18]. □

THEOREM 2. ATC-Problem *is NP-hard.*

PROOF. We reduce the WDalK-Problem to ATC-Problem. The complete proof is available in [18]. □

In view of the hardness, a natural question is whether efficient approximation algorithms can be designed for ATC-Problem. Thus, we investigate the properties of the problem in the next subsections. Observe that from the proof, it is apparent that the hardness comes mainly from maximizing the attribute score of a ATC.

5.2 Properties of (k, d)-truss

Our attribute truss community model is based on the concept of k-truss, so the communities inherit good structural properties of k-trusses, such as *k-edge-connected*, *bounded diameter* and *hierarchical structure*. In addition, since the attribute truss community is required to have a bounded query distance, it will have a small diameter, as explained below.

A k-truss community is $(k-1)$-edge-connected, since it remains connected whenever fewer than $k-1$ edges are deleted from the community [8]. Moreover, a k-truss based community has *hierarchical structure* that represents the hearts of the community at different levels of granularity [16], i.e., a k-truss is always contained in some $(k-1)$-truss . In addition, for a connected k-truss with n vertices, the diameter is at most $\lfloor \frac{2n-2}{k} \rfloor$ [8]. Small diameter is considered an important property of a good community [12].

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 community diameter in terms of the query distance as follows.

OBSERVATION 1. *For a* (k, d)-truss H *and a set of nodes* $V_q \subseteq H$*, we have* $d \leq \mathsf{diam}(H) \leq \min\{\frac{2|V(H)|-2}{k}, 2d\}$.

Note that another definition of dense subgraph, k-(r,s)-nucleus [31], is a generalized concept of k-truss, which can achieve very dense strucutre. However, whenever $s > 3$, finding k-(r,s)-nucleus is more expensive than computing k-trusses [18].

5.3 Properties of attribute score function

We next investigate the properties of the attribute score function, in search of prospects for an approximation algorithm for finding ATC. From the definition of attribute score function $\mathsf{f}(H, W_q)$, we can infer the following useful properties.

Positive influence of relevant attributes. The more relevant attributes a community H has, the higher the score $\mathsf{f}(H, W_q)$. E.g., consider the community H_4 and $W_q = \{ML\}$ in Figure 2 (d). If the additional attribute "ML" is added to the vertex v_9, then it can be verified that the score $\mathsf{f}(H_4, \{ML\})$ will increase. We have:

OBSERVATION 2. *Given a* ATC H *and a vertex* $v \in H$*, let a new input attribute* $w \in W_q \setminus \mathsf{attr}(v)$ *be added to* v*, and* H' *denote the resulting community. Then* $\mathsf{f}(H', W_q) > \mathsf{f}(H, W_q)$.

In addition, we have the following easily verified observation.

OBSERVATION 3. *Given a* ATC H *and query attribute sets* $W_q \subseteq W_{q'}$*, we have* $\mathsf{f}(H, W_q) \leq \mathsf{f}(H, W_{q'})$.

Negative influence of irrelevant vertices. Adding irrelevant vertices with no query attributes to a ATC will decrease its attribute score. For example, for $W_q = \{DB\}$, if we insert the vertex v_7 with attribute IR into the community H_1 in Figure 2(b), it decreases the score of the community w.r.t. the above query attribute $W_q = \{DB\}$, i.e., $\mathsf{f}(H_1 \cup \{v_7\}, \{DB\}) < \mathsf{f}(H_1, \{DB\})$. The following observation formalizes this property.

OBSERVATION 4. *Given two* ATC*'s* H *and* H' *where* $H \subset H'$*, suppose* $\forall v \in V(H') \setminus V(H)$ *and* $\forall w \in W_q$, $\mathsf{attr}(v) \cap V_w = \emptyset$. *Then* $\mathsf{f}(H', W_q) < \mathsf{f}(H, W_q)$.

Non-monotone property and majority attributes. The attribute score function is in general non-monotone w.r.t. the size of the community, even when vertices with query related attributes are added. For instance, for the community H_1 in Figure 2(a), with $W_q = \{DB, IR\}$, $\mathsf{f}(H_1, W_q) = 4 \cdot \frac{4}{4} = 4$. Let us add vertex v_7 with attribute IR into H_1 and represent the resulting graph as H_5, then $\mathsf{f}(H_5, W_q) = 4 \cdot \frac{4}{5} + 1 \cdot \frac{1}{5} = \frac{17}{5} < \mathsf{f}(H_1, W_q)$. If vertex v_7 has attribute DB instead of IR, then it is easy to verify that the attribute score of the resulting graph w.r.t. W_q is strictly higher than 4. Thus, $\mathsf{f}(.,.)$ is neither monotone nor anti-monotone. This behavior raises challenges for finding ATC with the maximum attibute score. Based on the above examples, we have the following observation.

OBSERVATION 5. *There exist* ATC*'s* H *and* H' *with* $V(H') = V(H) \cup \{v\}$*, and* $\mathsf{attr}(v) \cap W_q \neq \emptyset$*, such that* $\mathsf{f}(H', W_q) < \mathsf{f}(H, W_q)$*, and there exist* ATC*'s* H *and* H' *with* $V(H') = V(H) \cup \{v\}$*, and* $\mathsf{attr}(v) \cap W_q \neq \emptyset$*, for which* $\mathsf{f}(H', W_q) > \mathsf{f}(H, W_q)$.

The key difference between the two examples above is that DB is a "majority attribute" in H_1, a notion we formalize next. Formally, given a community H and query W_q, we say that a set of attributes X includes majority attributes of H, and $\theta(H, W_q \cap X) = \sum_{w \in W_q \cap X} \theta(H, w) \geq \frac{\mathsf{f}(H, W_q)}{2|V(H)|}$. Recall that $\theta(H, w)$ is the fraction of vertices of H containing the attribute w. We have:

LEMMA 1. *Let* H *be a* ATC *of a graph* G*. Suppose there is a vertex* $v \notin V(H)$ *such that the set of attributes* $W_q \cap \mathsf{attr}(v)$ *includes the majority attributes of* H *and that adding* v *to* H *results in a* ATC H' *of* G*. Then* $\mathsf{f}(H', W_q) > \mathsf{f}(H, W_q)$ *holds.*

PROOF. Suppose $W_q = \{w_1, ..., w_l\}$ and w.l.o.g., let $W_q \cap \mathsf{attr}(v) = \{w_1, ..., w_r\}$, where $1 \leq r \leq l$. Let $|V(H)| = b$, and for each attribute $w_i \in W_q$, let $|V(H) \cap V_{w_i}| = b_i$. Since $W_q \cap \mathsf{attr}(v)$ includes the majority attributes of H, $\theta(H, W_q \cap \mathsf{attr}(v)) = \frac{\sum_{i=1}^{r} b_i}{b} \geq \frac{\mathsf{f}(H,W_q)}{2b}$, so we have $\sum_{i=1}^{r} 2b_i \geq \mathsf{f}(H, W_q)$.

We have $\mathsf{f}(H, W_q) = \sum_{i=1}^{l} \frac{|V(H) \cap V_{w_i}|^2}{|V(H)|} = \sum_{i=1}^{l} \frac{b_i^2}{b}$, and $\mathsf{f}(H', W_q) = \sum_{i=1}^{r} \frac{(b_i+1)^2}{b+1} + \sum_{i=r+1}^{l} \frac{b_i^2}{b+1}$. As a result, $\mathsf{f}(H', W_q) - \mathsf{f}(H, W_q) = \frac{b \cdot \sum_{i=1}^{r}(2b_i+1) - \sum_{i=1}^{l} b_i^2}{b(b+1)} \geq \frac{b \cdot \mathsf{f}(H,W_q) + rb - b \cdot \mathsf{f}(H,W_q)}{b(b+1)} = \frac{r}{b+1} > 0$. ☐

This lemma will be helpful in designing bottom-up algorithms, by iteratively adding vertices with majority attributes to increase attribute score.

Non-submodularity and Non-supermodularity. A set function $g : 2^U \to \mathbb{R}^{\geq 0}$ is said to be submodular provided for all sets $S \subset T \subset U$ and element $x \in U \setminus T$, $g(T \cup \{x\}) - g(T) \leq g(S \cup \{x\}) - g(S)$, i.e., the marginal gain of an element has the so-called "diminishing returns" property. The function $g(.)$ is said to be supermodular if $-g(.)$ is submodular. Optimization problems over submodular functions lend themselves to efficient approximation. We thus study whether our attribute score function $f(.,.)$ is submodular w.r.t. its first argument, viz., set of vertices.

Consider the graph G in Figure 1 and query $W_q = \{DB, DM\}$ with $k = 2$. Let the induced subgraphs of G by the vertex sets $S_1 = \{q_1, v_4\}$ and $S_2 = \{q_1, v_4, v_5\}$ respectively be denoted G_1 and G_2; $G_1 \subseteq G_2$. Let v^* be a vertex not in G_2. Let us compare the marginal gains $\mathsf{f}(G_1 \cup \{v^*\}, W_q) - \mathsf{f}(G_1, W_q)$ and $\mathsf{f}(G_2 \cup \{v^*\}, W_q) - \mathsf{f}(G_2, W_q)$, from adding the new vertex v^* to G_1 and G_2. Suppose $v^* = v_6$ with attribute "DB", then we have $\mathsf{f}(G_2 \cup \{v_6\}, W_q) - \mathsf{f}(G_2, W_q) = (4 + 1/4) - (3 + 1/3) = 11/12 > \mathsf{f}(G_1 \cup \{v_6\}, W_q) - \mathsf{f}(G_1, W_q) = (3 + 1/3) - (2 + 1/2) = 5/6$, violating submodularity of the attribute score function $\mathsf{f}(.,.)$. On the other hand, suppose $v^* = q_2$ with attributes "DB" and "DM". Then we have $\mathsf{f}(G_2 \cup \{q_2\}, W_q) - \mathsf{f}(G_2, W_q) = (4+1) - (3+1/3) = 5/3 < \mathsf{f}(G_1 \cup \{q_2\}, W_q) - \mathsf{f}(G_1, W_q) = (3+4/3) - (2+1/2) = 11/6$, which violates supermodularity. We just proved:

LEMMA 2. *The attribute score function $\mathsf{f}(H, W_q)$ is neither submodular or supermodular.*

In view of this result, we infer that the prospects for an efficient approximation algorithm are not promising.

6. TOP-DOWN GREEDY ALGORITHM

In this section, we develop a greedy algorithmic framework for finding a ATC. It leverages the notions of attribute score contribution and attribute marginal gain that we define. Our algorithm first finds a (k, d)-truss, and then iteratively removes vertices with smallest attribute score contribution. Then, we analyze the time and space complexity of our algorithm. We also propose a more efficient algorithm with better quality, based on attribute marginal gain and bulk deletion.

6.1 Basic Algorithm

We begin with attribute score contribution. Given a subgraph $H \subset G$, a vertex $v \in V(H)$, and attribute query W_q, let us examine the change to the score $f(H, W_q)$ from dropping v.

$$
\begin{aligned}
\mathsf{f}(H - \{v\}, W_q) &= \sum_{w \in W_q} \frac{|V_w \cap (V(H) - \{v\})|^2}{|V(H)| - 1} \\
&= \sum_{w \in W_q - \mathsf{attr}(v)} \frac{|V_w \cap V(H)|^2}{|V(H)| - 1} + \sum_{w \in W_q \cap \mathsf{attr}(v)} \frac{(|V_w \cap V(H)| - 1)^2}{|V(H)| - 1} \\
&= \sum_{w \in W_q} \frac{|V_w \cap V(H)|^2}{|V(H)| - 1} - \sum_{w \in \mathsf{attr}(v) \cap W_q} \frac{|V_w \cap V(H)|^2}{|V(H)| - 1} + \\
&+ \sum_{w \in W_q \cap \mathsf{attr}(v)} \frac{(|V_w \cap V(H)| - 1)^2}{|V(H)| - 1} \\
&= \sum_{w \in W_q} \frac{|V_w \cap V(H)|^2}{|V(H)| - 1} - \sum_{w \in W_q \cap \mathsf{attr}(v)} \frac{2|V_w \cap V(H)| - 1}{|V(H)| - 1} \\
&= \frac{\mathsf{f}(H, W_q) \cdot |V(H)|}{|V(H)| - 1} - \frac{\sum_{w \in W_q \cap \mathsf{attr}(v)} (2|V_w \cap V(H)| - 1)}{|V(H)| - 1}
\end{aligned}
$$

The second term represents the drop in the attribute score of H from removing v. We would like to remove vertices with the least drop in score. This motivates the following.

DEFINITION 7 (ATTRIBUTE SCORE CONTRIBUTION). *Given a graph H and attribute query W_q, the attribute score contribution of a vertex $v \in V(H)$ is defined as $\mathsf{f}_H(v, W_q) = \sum_{w \in W_q \cap \mathsf{attr}(v)} 2|V_w \cap V(H)| - 1$.*

The intuition behind dropping a vertex v from H is as follows. Since $\mathsf{f}(H, W_q)$ is non-monotone (Section 5.3), the updated score from dropping v from H may increase or decrease, so we check if $\mathsf{f}(H - v, W_q) > \mathsf{f}(H, W)$.

Algorithm overview. Our first greedy algorithm, called Basic, has three steps. First, it finds the maximal (k, d)-truss of G as a candidate. Second, it iteratively removes vertices with smallest attribute score contribution from the candidate graph, and maintains the remaining graph as a (k, d)-truss, until no longer possible. Finally, it returns a (k, d)-truss with the maximum attribute score among all generated candidate graphs as the answer.

The details of the algorithm follow. First, we find the maximal (k, d)-truss of G as G_0. Based on the given d, we compute a set of vertices S having query distance no greater than d, i.e., $S_0 = \{u : \mathsf{dist}_G(u, Q) \le d\}$. Let $G_0 \subset G$ be the subgraph of G induced by S_0. Since G_0 may contain edges with support $< (k-2)$, we invoke the following steps to prune G_0 into a (k, d)-truss.

Algorithm 1 Basic (G, Q)

Input: A graph $G = (V, E)$, a query $Q = (V_q, W_q)$, numbers k and d.
Output: A (k, d)-truss H with the maximum $\mathsf{f}(H, W_q)$.

```
Find a set of vertices S_0 having the query distance ≤ d, i.e., S_0 = {u :
   dist_G(u, Q) ≤ d}.
Let G_0 be the induced subgraph of S, i.e., G_0 = (S_0, E(S_0)), where
   E(S_0) = {(v, u) : v, u ∈ S_0, (v, u) ∈ E}.
Maintain G_0 as a (k, d)-truss.
Let l ← 0;
while connect_{G_l}(Q) = true do
    Compute the attribute score of f(G_l, W_q);
    Compute f_{G_l}(u, W_q) by Def. 7;
    u* ← arg min_{u∈V(G_l)−V_q} f_{G_l}(u, W_q);
    Delete u* and its incident edges from G_l;
    Maintain G_l as a (k, d)-truss.
    G_{l+1} ← G_l; l ← l + 1;
H ← arg max_{G'∈{G_0,...,G_{l−1}}} f(G', W_q);
```

(k, d)**-truss maintenance**: repeat until no longer possible:
(i) *k-truss*: remove edges contained in $< (k-2)$ triangles;
(ii) *query distance*: remove vertices with query distance $> d$, and their incident edges;

Notice that the two steps above can trigger each other: removing edges can increase query distance and removing vertices can reduce edge support. In the following, we start from the maximal (k, d)-truss G_l where $l = 0$, and find a (k, d)-truss with large attribute score by deleting a vertex with the smallest attribute score contribution.

Finding a (k, d)-truss with large attribute score. G_0 is our first candidate answer. In general, given G_l, we find a vertex $v \in V(G_l) \setminus V_q$ with the smallest attribute score contribution and remove it from G_l. Notice that v cannot be one of the query vertices. The removal may violate the (k, d)-truss constraint so we invoke the (k, d)-truss maintenance procedure above to find the next candidate answer. We repeat this procedure until G_l is not a (k, d)-truss any more. Finally, the candidate answer with the maximum attribute score generated during this process is returned as the final answer, i.e., $\arg\max_{G' \in \{G_0, \ldots, G_{l-1}\}} \mathsf{f}(G', W_q)$. The detailed description is presented in Algorithm 1.

EXAMPLE 2. *We apply Algorithm 1 on the graph G in Figure 1 with query $Q = (\{q_1\}, \{DB, DM\})$, for $k = 4$ and $d = 2$. First, the algorithm finds the (k, d)-truss G_0 as the subgraph H shown in Figure 1. Next, we select vertex v_7 with the minimum attribute score contribution $\mathsf{f}_{G_0}(v_7, W_q) = 0$ and remove it from G_0. Indeed it contains neither of the query attributes. Finally, the algorithm finds the ATC H_2 with the maximum attribute score in Figure 2(b), which, for this example, is the optimal solution.*

6.2 Complexity Analysis

In each iteration i of Algorithm 1, we delete at least one vertex and its incident edges from G_i. Clearly, the number of removed edges is no less than $k-1$, and so the total number of iterations is $t \le \min\{n-k, m/(k-1)\}$. We have:

THEOREM 3. *Algorithm 1 takes $O(m\rho + t(|W_q|n + |V_q|m))$ time and $O(m + |\mathsf{attr}(V)|)$ space, where $t \in O(\min\{n, m/k\})$, and ρ is the arboricity of graph G with $\rho \le \min\{d_{max}, \sqrt{m}\}$.*

Proof Sketch: The time cost of Algorithm 1 mainly comes from three key parts: (1) query distance computation takes $O(t|V_q|m)$

time by BFS traversals; (2) k-truss maintenance takes $O(\rho \cdot m)$ time; (3) attribute score computation takes $O(t|W_q| \cdot n)$ time. A complete proof is available in [18]. It was shown in [7] that $\rho \leq \min\{d_{max}, \sqrt{m}\}$. $\square$

6.3 An improved greedy algorithm

The greedy removal strategy of Basic is simple, but suffers from the following limitations on quality and efficiency. Firstly, the attribute score contribution myopically considers the removal vertex v only, and ignores its impact on triggering removal of other vertices, due to violation of k-truss or distance constraints. If these vertices have many query attributes, it can severely limit the effectiveness of the algorithm. Thus, we need to look ahead the effect of each removal vertex, and then decide which ones are better to be deleted. Secondly, Basic removes only one vertex from the graph in each step, which leads to a large number of iterations, making the algorithm inefficient.

In this section, we propose an improved greedy algorithm called BULK, which is outlined in Algorithm 2. BULK uses the notion of attribute marginal gain and a bulk removal strategy.

Attribute Marginal Gain. We begin with a definition.

DEFINITION 8 (ATTRIBUTE MARGINAL GAIN). *Given a graph H, attribute query W_q, and a vertex $v \in V(H)$, the attribute marginal gain is defined as* $\mathsf{gain}_H(v, W_q) = \mathsf{f}(H, W_q) - \mathsf{f}(H - S_H(v), W_q)$*, where $S_H(v) \subset V(H)$ is v together with the set of vertices that violate* (k, d)-truss *after the removal of v from H.*

Notice that by definition, $v \in S_H(v)$. For example, consider the graph G in Figure 1 and the query $Q = (\{q_1\}, \{ML\})$, with $k = 3$ and $d = 2$. The vertex v_9 has no attribute "ML", and the attribute score contribution is $\mathsf{f}_G(v_9, W_q) = 0$ by Definition 7, indicating no attribute score contribution by vertex v_9. However, the fact is that v_9 is an important bridge for connections among the vertices q_1, v_8, and v_{10} with attribute "ML". The deletion of v_9 will thus lead to the deletion of v_8 and v_{10}, due to the 3-truss constraint. Thus, $S_G(v_9) = \{v_8, v_9, v_{10}\}$. The marginal gain of v_9 is $\mathsf{gain}_G(v_9, W_q) = \mathsf{f}(G, W_q) - \mathsf{f}(G - S_G(v_9), W_q) = \frac{3}{4} - \frac{1}{9} > 0$. This shows that the deletion of v_9 from G decreases the attribute score. It illustrates that attribute marginal gain can more accurately estimate the effectiveness of vertex deletion than score attribute contribution, by naturally incorporating look-ahead.

One concern is that $\mathsf{gain}_H(v, W_q)$ needs the exact computation of $S_H(v)$, which has to simulate the deletion of v from H by invoking (k, d)-truss maintenance, which is expensive. An important observation is that if vertex v is to be deleted, its neighbors $u \in N(v)$ with degree $k-1$ will also be deleted, to maintain k-truss. Let $P_H(v)$ be the set of $v's$ 1-hop neighbors with degree $k-1$ in H, i.e., $P_H(v) = \{u \in N(v) : \deg_H(u) = k-1\}$. We propose a local attribute marginal gain, viz., $\hat{\mathsf{gain}}_H(v, W_q) = \mathsf{f}(H, W_q) - \mathsf{f}(H - P_H(v), W_q)$, to approximate $\mathsf{gain}_H(v, W_q)$. Continuing with the above example, in graph G, for deleting vertex v_9, note that $\deg(v_8) = \deg(v_{10}) = 2 = k-1$, so we have $P_G(v_9) = \{v_8, v_9, v_{10}\}$, which coincides with $S_G(v_9)$. In general, $\hat{\mathsf{gain}}_H(v, W_q)$ serves as a good approximation to $\mathsf{gain}_H(v, W_q)$ and can be computed more efficiently.

Bulk Deletion. The second idea incorporated in BULK is bulk deletion. The idea is that instead of removing one vertex with the smallest attribute marginal gain, we remove a small percentage of vertices from the current candidate graph that have the smallest attribute marginal gain. More precisely, let G_i be the current candidate graph and let $\epsilon > 0$. We identify the set of vertices S such that

---

Algorithm 2 BULK (G, Q)

Input: A graph $G = (V, E)$, a query $Q = (V_q, W_q)$, numbers k and d, parameter ε.
Output: A (k, d)-truss H with the maximum $\mathsf{f}(H, W_q)$.

Find the maximal (k, d)-truss G_0.
Let $l \leftarrow 0$;
while $\mathsf{connect}_{G_l}(Q) =$ **true do**
  Find a set of vertices S of the smallest $\hat{\mathsf{gain}}_{G_l}(v, W_q)$ with the size of $|S| = \frac{\varepsilon}{1+\varepsilon}|V(G_i)|$;
  Delete S and their incident edges from G_l;
  Maintain the (k, d)-truss of G_l;
  $G_{l+1} \leftarrow G_l$; $l \leftarrow l + 1$;
$H \leftarrow \arg\max_{G' \in \{G_0, \ldots, G_{l-1}\}} \mathsf{f}(G', W_q)$;

$|S| = \frac{\epsilon}{1+\epsilon}|V(G_i)|$ and the vertices in S have the smallest attribute marginal gain, and remove S from G_i .

7. INDEX-BASED SEARCH ALGORITHM

While the BULK algorithm based on the framework of Algorithm 1 has polynomial time complexity, when the graph G is large and the query Q has many attributes, finding ATCs entails several ATC queries, which can be expensive. To help efficient processing of ATC queries, we propose a novel index called attributed-truss index (ATindex). It maintains known graph structure and attribute information.

7.1 Attributed Truss Index

The ATindex consists of two components: *structural trussness* and *attribute trussness*.

Structural Trussness. Recall that trusses have a hierarchical structure, i.e., for $k \geq 3$, a k-truss is always contained in some $(k-1)$-truss [16]. For any vertex or any edge, there exists a k-truss with the largest k containing it. We define the trussness of a subgraph, an edge, and a vertex as follows.

DEFINITION 9 (TRUSSNESS). *Given a subgraph $H \subseteq G$, the trussness of H 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 $\tau_G(e) = \max_{H \subseteq G \wedge e \in E(H)}\{\tau(H)\}$. The trussness of a vertex $v \in V(G)$ is $\tau_G(v) = \max_{H \subseteq G \wedge v \in V(H)}\{\tau(H)\}$.*

Consider the graph G in Figure 1, the trussness of the edge $e(q_1, v_1)$ is 4, because there exists a 4-truss containing $e(q_1, v_1)$ in Figure 2(b), and any subgraph H containing $e(q_1, v_1)$ has $\tau(H) \leq 4$, i.e., $\tau_G(e(q_1, v_1)) = \max_{H \subseteq G \wedge e \in E(H)}\{\tau(H)\} = 4$. Based on the trussness of a vertex (edge), we can infer in constant time whether there exists a k-truss containing it. Notice that for a graph G, $\bar{\tau}(\emptyset)$ denotes the maximum structural trussness of G.

Attributed Trussness. Structural trussness index is not sufficient for ATC queries. Given a vertex v in G with structural trussness $\tau_G(v) \geq k$, there is no guarantee that v will be present in a (k, d)-truss with large attribute score w.r.t. query attributes. E.g., consider the graph G and vertex v_1 with $\tau_G(v_1) = 4$ in Figure 1. Here, v_1 will not be present in an ATC for query attributes $W_q = \{\text{"}ML\text{"}\}$ since it does not have attribute "ML". On the contrary, v_1 is in a ATC w.r.t. $W_q = \{\text{"}DM\text{"}\}$. By contrast, v_9 is *not* present in a 4-truss w.r.t. attribute "DM" even though it has that attribute. To make such searches efficient, for each attribute $w \in \mathcal{A}$, we consider an attribute projected graph, which only contains the vertices associated with attribute w, formally defined below.

DEFINITION 10. ***(Attribute Projected graph & Attributed Trussness).*** *Given a graph G and an attribute $w \in A(V)$, the projected*

Algorithm 3 LocATC (G, Q)

Input: A graph $G = (V, E)$, a query $Q = (V_q, W_q)$.
Output: A (k, d)-truss H with the maximum $\mathsf{f}(H, W_q)$.

1: Compute an attribute Steiner tree T connecting V_q using attribute truss distance as edge weight;
2: Iteratively expand T into graph G_t by adding adjacent vertices v, until $|V(G_t)| > \eta$;
3: Compute a connected k-truss containing V_q of G_t with the largest trussness $k = k_{max}$;
4: Let the k_{max}-truss as the new G_t.
5: Apply Algorithm 2 on G_t to identify ATC with parameters $k = k_{max}$ and $d = \mathsf{dist}_{G_t}(G_t, V_q)$.

graph of G on attribute w is the induced subgraph of G by V_w, i.e., $G_w = (V_w, E_{V_w}) \subseteq G$. Thus, for each vertex v and edge e in G_w, the attributed trussness of v and e w.r.t. w in G_w are respectively defined as $\tau_{G_w}(v) = \max_{H\subseteq G_w \wedge v\in V(H)} \{\tau(H)\}$ and $\tau_{G_w}(e) = \max_{H\subseteq G_w \wedge e\in E(H)}\{\tau(H)\}$.

For instance, for the graph G in Figure 1, the projected graph G_w of G on w = "DB" is the graph H_1 in Figure 2(a). For vertices v_1 and v_4, even though both have the same structural trussness $\tau_G(v_1) = \tau_G(v_4) = 4$, in graph H_1, vertex v_4 has attribute trussness $\tau_{H_1}(v_4) = 4$ w.r.t. w = "DB", whereas vertex v_1 is not even present in H_1, indicating that vertex v_4 is more relevant with "DB" than v_1.

7.2 Index-based Query Processing

In this section, we propose an ATindex-based query processing algorithm by means of local exploration, called LocATC.

Algorithm overview. Based on the ATindex, the algorithm first efficiently detects a small neighborhood subgraph around query vertices, which tends to be densely and closely connected with the query attributes. Then, we apply Algorithm 2 to shrink the candidate graph into a (k, d)-truss with large attribute score. The outline of the algorithm LocATC is presented in Algorithm 3. Note that, when no input parameters k and d are given in LocATC, we design an auto-setting mechanism for parameters k and d, which will be explained in Section 8.

To find a small neighborhood candidate subgraph, the algorithm starts from the query vertices V_q, and finds a Steiner tree connecting the query vertices. It then expands this tree by adding attribute-related vertices to the graph. Application of standard Steiner tree leads to poor quality, which we next explain and address.

Finding attributed Steiner tree T. As discussed above, a Steiner tree connecting query vertices is used as a seed for expanding into a (k, d)-truss. A naive method is to find a minimal weight Steiner tree to connect all query vertices, where the weight of a tree is the number of edges. Even though the vertices in such a Steiner tree achieve close distance to each other, using this tree seed may produce a result with a small trussness and low attribute score. For example, for the query $Q = (\{q_1, q_2\}, \{DB\})$ (see Figure 1), the tree $T_1 = \{(q_1, v_1)\ , (v_1, q_2)\}$ achieves a weight of 2, which is optimal. However, the edges (q_1, v_1) and (v_1, q_2) of T_1 will not be present in any 2-truss with the homogeneous attribute of "DB". Thus it suggests growing T_1 into a larger graph will yield a low attribute score for W_q = "DB". On the contrary, the Steiner tree T_2 = $\{(q_1, v_4)\ , (v_4, q_2)\}$ also has a total weight of 2, and both of its edges have the attribute trussness of 4 w.r.t. the attribute "DB", indicating it could be expanded into a community with large attribute score. For discriminating between such Steiner trees, we propose a notion of attributed truss distance.

DEFINITION 11 (ATTRIBUTE TRUSS DISTANCE). *Given an edge $e = (u, v)$ in G and query attributes W_q, let $\mathcal{G} = \{G_w : w \in W_q\} \cup \{G\}$. Then the attribute truss distance of e is defined as $\hat{\mathsf{dist}}_{W_q}(e) = 1 + \gamma(\sum_{g\in\mathcal{G}}(\bar{\tau}(\emptyset) - \tau_g(e)))$, where $\bar{\tau}(\emptyset)$ is the maximum structural trussness in graph G.*

The set $\mathcal{G}$ consists of G together with all its attribute projected graphs G_w, for $w \in W_q$ and the difference $(\bar{\tau}(\emptyset) - \tau_g(e))$ measures the shortfall in the attribute trussness of edge e w.r.t. the maximum trussness in G. The sum $\sum_{g\in\mathcal{G}}(\bar{\tau}(\emptyset) - \tau_g(e))$ indicates the overall shortfall of e across G as well as all its attribute projections. Smaller the shortfall of an edge, lower its distance. Finally, γ controls the extent to which small value of structural and attribute trussness, i.e., a large shortfall, is penalized. Using ATindex, for any edge e and any attribute w, we can access the structural trussness $\tau_G(e)$ and attribute trussness $\tau_{G_w}(e)$ in $O(1)$ time. Since finding minimum weight Steiner tree is NP-hard, we apply the well-known algorithm of [23, 28] to obtain a 2-approximation, using attributed truss distance. The algorithm takes $O(m|W_q| + m + n\log n) \subseteq O(m|W_q| + n\log n)$ time, where $O(m|W_q|)$ is the time taken to compute the attributed truss distance for m edges.

Expand attribute Steiner tree T to Graph G_t. Based on the attribute Steiner tree T built above, we locally expand T into a graph G_t as a candidate (k, d)-truss with numerous query attributes. Recall that Lemma 1 gives a useful principle to expand the graph with insertion of a vertex at a time, while increasing the attribute score. Specifically, if $\theta(G_t, W_q \cap \mathsf{attr}(v)) \geq \frac{\mathsf{f}(G_t, W_q)}{2|V(G_t)|}$, then graph $G_T \cup \{v\}$ has a larger attribute score than G_T. We can identify such vertices whose attribute set includes majority attributes of the current candidate graph and add them to the current graph.

Now, we discuss the expansion process, conducted in a BFS manner. We start from vertices in T, and iteratively insert adjacent vertices with the largest vertex attribute scores into G_t until the vertex size exceeds a threshold η, i.e., $|V(G_t)| \leq \eta$, where η is empirically tuned. After that, for each vertex $v \in V(G_t)$, we add all its adjacent edges e into G_t.

Apply BULK on G_t with auto-setting parameters. Based on the graph G_t constructed above, we apply Algorithm 2 with given parameters k and d on G_t to find an ATC. If input parameters k and d are not supplied, we can set them automatically as follows. We first compute a k-truss with the largest k connecting all query vertices. Let k_{max} denote the maximum trussness of the subgraph found. We set the parameter k to be k_{max}. We also compute the query distance of G_t and assign it to d, i.e., $d := \mathsf{dist}_{G_t}(G_t, V_q)$. We then invoke the BULK algorithm on G_t with parameters k, d to obtain a ATC with large trussness and high attribute cohesiveness.

Friendly mechanism for query formulation. Having to set values for many parameters for posing queries using LocATC can be daunting. To mitigate this, we make use of the auto-setting of parameters k and d. Additionally, we allow the user to omit the query attribute parameter W_q in a query $Q(V_q, W_q)$ and write $Q(V_q, \_)$. Thus, only query nodes need to be specified. Our algorithm will automatically set $W_q := \bigcup_{v\in V_q} A(v)$ by default. The rationale is that the algorithm will take the whole space of all possible attributes as input, and leverage our community search algorithms to find communities with a proper subspace of attributes, while achieving high scores. For example, consider the query $Q = (\{q_1, q_2\}, \_)$ on graph G in Figure 1, LocATC automatically sets $W_q := \{DB, DM, ML\}$. The discovered community is shown in Figure 2(b), which illustrates the feasibility of this strategy. This auto-complete mechanism greatly facilitates query formulation to identify relative attributes for discovered communities, which benefits users in a simple way.

Handling bad queries. In addition to auto-complete query formulation, we discuss how to handle bad queries issued by users. Bad queries contain query nodes and query attributes that do not constitutes a community. Our solution is to detect outliers of bad queries and then suggest good candidate queries for users. The whole framework includes three steps. (1) It identifies bad queries. Based on the structural constraint of (k, d)-truss, if query nodes span a long distance and are loosely connected in the graph, the query tends to be a bad query. In addition, if none of the query attributes are present in the proximity of query nodes, it suggests there are no communities with homogeneous attributes, from among the query attributes, thus indicating the query is bad. (2) It recommends candidates for good queries. Due to outliers existing in bad queries, we partition the given query into several small queries. Based on the distribution of graph distance, graph cohesiveness, and query attribute, we partition given query nodes into several disjoint good queries. (3) Whenever a query does not admit a (k, d)-truss containing query nodes or does not lead to a (k, d)-truss with attributes relevant to the query, our algorithm can quickly detect this and return an empty answer. See [18] for experiments on handling bad queries.

8. EXPERIMENTS

In this section, we test all proposed algorithms on a Linux Server with Intel Xeon CUP X5570 (2.93 GHz) and 50GB main memory.

8.1 Experimental Setup

Datasets. We conduct experimental studies using 9 real-world networks with ground-truth communities.The network statistics are reported in Table 2.

The preceding 4 datasets, Krogan, Facebook, Cornell, and Texas, are real-world datasets with real attributes. Krogan is one PPI network related to the yeast Saccharomyces cerevisiae [15]. The second dataset is Facebook ego-networks [27]. For a given user id X in Facebook network G, the ego-network of X, denoted ego-facebook-X, is the induced subgraph of G by X and its neighbors. Facebook dataset contains contains 10 ego-networks indicated by its ego-user X, where $X \in \{0, 107, 348, 414, 686, 698, 1684, 1912, 3437, 3890\}$. For simplicity, we abbreviate ego-facebook-X to fX, e.g., f698. Cornell and Texas are web graphs [1].

The other 5 networks, Amazon, DBLP, Youtube, LiveJournal and Orkut, contain 5000 top-quality ground-truth communities. However, since the vertices on these networks have no attributes, we generate an attribute set consisting of $|\mathcal{A}| = 0.005 \cdot |V|$ different attribute values in each network G. The average number of attribute/vertex $\frac{|\mathcal{A}|}{|V|} = 0.005$ is less than the proportion of attributes to vertices in datasets with real attributes (e.g., the value of 0.12 in Facebook) in Table 2. A smaller attribute pool $\mathcal{A}$ makes homogeneity of synthetic attributes in different communities more likely, which stresses testing our algorithms. For each ground-truth community, we randomly select 3 attributes, and assign each of these attributes to each of random 80% vertices in the community. In addition, to model noise in the data, for each vertex in the graph, we randomly assign a random integer of $[1, 5]$ attributes to it. Except Krogan, Cornell, and Texas, all other datasets are available from the Stanford Network Analysis Project.[2]

[1] http://linqs.cs.umd.edu/projects/projects/lbc/
[2] snap.stanford.edu

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

Network	$\|V\|$	$\|E\|$	d_{max}	$\bar{\tau}(\emptyset)$	$\|\mathcal{A}\|$	$\|\mathsf{attr}(V)\|$
Krogan	2.6**K**	7.1**K**	140	16	3064	28151
Facebook	1.9**K**	8.9**K**	416	29	228	3944
Cornell	195	304	94	4	1588	18496
Texas	187	328	104	4	1501	15437
Amazon	335**K**	926**K**	549	7	1674	1804406
DBLP	317**K**	1**M**	342	114	1584	1545490
Youtube	1.1**M**	3 **M**	28,754	19	5327	2163244
LiveJournal	4**M**	35**M**	14,815	352	11104	12426432
Orkut	3.1**M**	117**M**	33,313	78	9926	10373866

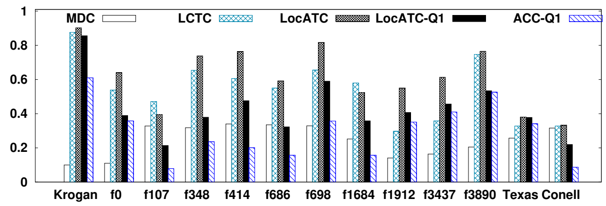


Figure 3: Quality evaluation (F_1 score) on networks with real attributes and ground-truth communities

Algorithms Compared. We test our three algorithms – Basic, BULK, and LocATC. Here, Basic is the Algorithm 1 that removes single node in each iteration. BULK is an improved greedy algorithm in Algorithm 2, which removes a set of nodes with size $\frac{\epsilon}{1+\epsilon}|V(G_i)|$ from graph G_i in each iteration. We empirically set $\epsilon = 0.03$. LocATC is the bottom-up local exploration approach in Algorithm 3. For all methods, we set the parameter $k = 4$ and $d = 4$ by default.

In addition, to evaluate the effectiveness of the ATC model on attributed graphs, we implemented three state-of-the-art community search methods – ACC [13], MDC [32] and LCTC [19]. Note that both MDC and LCTC only consider the graph structure and ignore the attributes. ACC considers both graph structure and attributes, but it only deals with a single query node with query attributes.

Queries. For each dataset, we randomly test 100 sets of queries $Q = (V_q, W_q)$, where we set both the number of query nodes $|V_q|$, and the number of query attributes $|W_q|$ to 2 by default.

Evaluation Metrics. To evaluate the quality of communities found by all algorithms, we measure the F1-score reflecting the alignment between a discovered community C and a ground-truth community $\hat{C}$. Given a ground-truth community $\hat{C}$, we randomly pick query vertices and query attributes from it and query the graph using different algorithms to obtain the discovered community C. Then, $F1$ 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. For all efficiency experiments, we consistently report the running time in seconds.

8.2 Quality and Efficiency Evaluation

To evaluate the effectiveness and efficiency of different community models, we compare LocATC with three state-of-the-art methods – ACC, MDC and LCTC on attributed networks with ground-truth communities.

Networks with real-world attributes. We experiment with the Krogan, Cornell, Texas, and the 10 Facebook ego-networks, all having real-world attributes. For every ground-truth community, we randomly select a set of query nodes with size drawn uniformly at random from $[1, 16]$. We use 2 representative attributes from the community as query attributes. We choose attributes occurring most frequently in a given community and rarely occurring in other communities as representative attributes. We evaluate the accuracy of detected communities and report the averaged F1-score over all

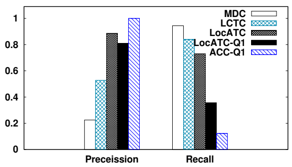


Figure 4: Comparison of precision and recall on f414 network.

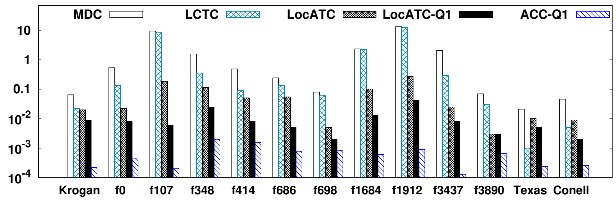
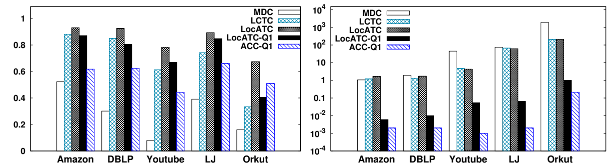


Figure 5: Efficiency evaluation (query time in seconds) on networks with real attributes and ground-truth communities



(a) F_1 score (b) Query time (in seconds)

Figure 6: Evaluation on networks with synthetic attributes and ground-truth communities

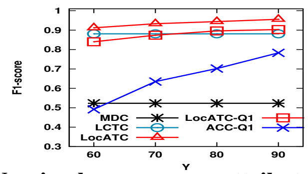


Figure 7: Varying homogeneous attributes on Amazon

queries on each network.

Figure 3 shows the F1-score results. Our method (LocATC) achieves the highest F1-score on most networks, except for facebook ego-networks f104 and f1684. The reason is that vertices of ground-truth communities in f104 and f1684 are strongly connected in structure, but are not very homogeneous on query attributes. MDC and LCTC do not perform as well as LocATC, because those community models only consider structure metrics, and ignore attribute features. Note that for each query with multiple query vertices, the attributed community search method ACC randomly takes one query vertex as input. We make this explicit and denote it as ACC-Q1 in Figure 3. For comparison, we apply the same query on our method LocATC, and denote it as LocATC-Q1. LocATC-Q1 clearly outperforms ACC-Q1 in terms of F1-score, showing the superiority of our ATC model. In addition, LocATC achieves higher score than LocATC-Q1, indicating our method can discover more accurate communities with more query vertices. Furthermore, we also compare the precision and recall of all methods on f414 network in Figure 4. MDC perform the worst on precision, since it considers no query attributes and includes many nodes that are not in ground-truth communities. ACC-Q1 is the winner on precision, which is explained by the strict attribute constraints in its definition. On the other hand, in terms of recall, ACC-Q1 is the worst method as it only identifies a small part of ground-truth communities. Overall, LocATC achieves a good balance between precision and recall. This is also reflected in LocATC achieving the best F1-score on most datasets (Figure 3).

Figure 5 shows the running time performance of all methods. In terms of supporting multiple query vertices, LocATC runs up to two orders of magnitude faster than MDC and LCTC on ego-networks in Facebook, and LCTC is the winner on Cornell and Texas networks. For one query vertex, ACC-Q1 runs faster than LocATC-Q1, since k-cores can be computed quicker than k-trusses.

Networks with synthetic attributes. In this experiment, we test on 5 large networks – DBLP, Amazon, Youtube, LiveJournal, and Orkut, with ground-truth communities and synthetic attributes [35]. Following the testing procedures of query generation and F1-score evaluation above, we observe similar results on datasets with synthetic attributes in Figure 6 (a) and (b).

8.3 Parameter Sensitivity Evaluation

In this experiment, we vary various parameters used in the synthetic data generation, query generation, and algorithm definitions, and evaluate the quality and efficiency performance of LocATC.

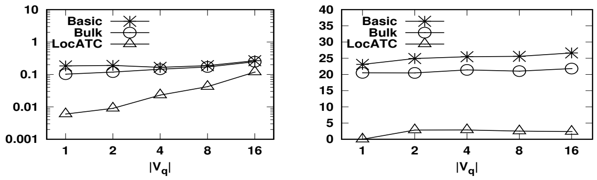
Varying homogeneous attributes in synthetic datasets. For each ground-truth community in Amazon, we randomly select 3 attributes, and assign each of these attributes to each of $Z\%$ vertices in the community, where Z is a random number in [50, Y]. Note that different attributes may have different values of Z. The parameter Y is varied from 60 to 90. As Y is increased, intuitively the level of homogeneity in the network and in its communities increases. The results of F1-score are shown in Figure 7. As homogeneous attributes in communities increase, MDC and LCTC maintain the same F1-score, while the F1-score of all methods of attributed community search – LocATC, LocATC-Q1, and ACC-Q1 – increases as homogeneity increases. Once again, LocATC is the best method even when the proportion of homogeneous attributes falls in [50, 60]. LocATC-Q1 beats ACC-Q1 for all settings of homogeneity. Similar results can be also observed on other synthetic datasets.

Varying query vertex size $|V_q|$. We evaluate the various approaches using different queries on f414 and DBLP. We test 5 different values of $|V_q|$, i.e., $\{1, 2, 4, 8, 16\}$. For each value of $|V_q|$, we randomly generate 100 sets of queries, and report the average running time. The results for f414 and DBLP are respectively shown in Figure 8 (a) and (b). LocATC achieves the best performance, and increases smoothly with the increasing query vertex size. BULK is more effective than Basic, thanks to the bulk deletion strategy. Most of the cost of BULK and Basic comes from computing the maximal (k, d)-truss G_0. All methods take less time on f414 than on DBLP network, due to the small graph size of f414.

Varying parameters ϵ. We test the performance of LocATC by varying ϵ. We used the same query nodes that are selected in Sec. 8.2 on f414 network. The results of F1-score and query time by varying ϵ are respectively reported in Figure 9 (a) and (b). As we can see, LocATC removes a smaller portion of nodes, which achieves a higher F1-score using more query time.

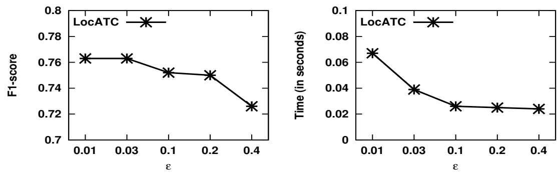
8.4 Case Study on PPI network

Besides the quality evaluation, we also apply LocATC on the protein-protein interaction (PPI) network Krogan to discover the protein complexes and investigate biologically significant clues. Figure 10(a) shows one complex "transcription factor TFIIIC complex" in sccharomyces cerevisiae, which is identified by biologists previously. The graph contains 6 nodes and 12 edges. Similar with the procedure of good query generation in Sec. 8.2, we randomly sample a query as $Q = (V_q, W_q)$ where V_q ={854277, 856100} and W_q ={"GO:0001009", "GO:0001041"}, and set the parameters $k = 3$ and $d = 3$. To illustrate the importance of the consideration of protein attributes in detecting protein complexes, we simply use the structure and find the $(3, 3)$-truss shown in Figure 10(b). This community contains 11 proteins including 6



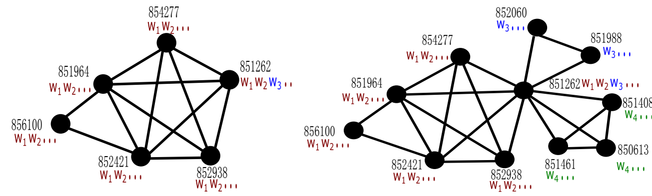
(a) f414 (b) DBLP

Figure 8: Varying query vertex size $|V_q|$: Query Time



(a) F1-score (b) Query Time

Figure 9: Varying ϵ on f414



(a) ATC & a protein complex (b) (3,3)-truss

Figure 10: $Q = (\{q_1, q_2\}, \{w_1, w_2\})$ where $q_1 = 854277$, $q_2 = 856100$ and $w_1 =$ "GO:0001009", $w_2 =$"GO:0001041"

proteins of the ground-truth complex of Figure 10(a). The other 5 proteins not present in the ground-truth complex are associated with no query attributes, but have other attributes w_3 and w_4, as shown in Figure 10(b). When we look up the database of Gene Ontology[3], we know that the attributes of "biological processes" as "GO:0001009" and "GO:0001041" respectively represent "transcription from RNA polymerase III hybrid type promoter" and "transcription from RNA polymerase III type 2 promoter". LocATC *is able to identify all proteins that perform the same biological process of transcription from RNA polymerase.* Overall, LocATC successfully identifies all proteins that constitute the ground-truth complex in Figure 10(a).

9. CONCLUSION

In this work, we propose an attributed truss community (ATC) model that allows to find a community containing query nodes with cohesive and tight structure, also sharing homogeneous query attributes. The problem of finding an ATC is *NP*-hard. We also show that the attribute score function is not monotone, submodular, or supermodular, indicating approximation algorithms may not be easy to find. We propose several carefully designed strategies to quickly find high-quality communities. We design an elegant and compact index, ATindex, and implement an efficient query processing algorithm, which exploits local exploration and bulk deletion. Extensive experiments reveal that our model and algorithms significantly outperform previous approaches.

Acknowledgment

Research was supported in part by a discovery and a discovery accelerator supplement grant from NSERC.

10. REFERENCES

[1] S. Agrawal, S. Chaudhuri, and G. Das. Dbxplorer: A system for keyword-based search over relational databases. In *ICDE*, pages 5–16, 2002.

[2] B. Bahmani, R. Kumar, and S. Vassilvitskii. Densest subgraph in streaming and mapreduce. *PVLDB*, 5(5):454–465, 2012.

[3] N. Barbieri, F. Bonchi, E. Galimberti, and F. Gullo. Efficient and effective community search. *DMKD*, 29(5):1406–1433, 2015.

[4] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using banks. In *ICDE*, pages 431–440, 2002.

[5] C. Bothorel, J. D. Cruz, M. Magnani, and B. Micenkova. Clustering attributed graphs: models, measures and methods. *Network Science*, 3(03):408–444, 2015.

[6] H. Cheng, Y. Zhou, X. Huang, and J. X. Yu. Clustering large attributed information networks: an efficient incremental computing approach. *DMKD*, 25(3):450–477, 2012.

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

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

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

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

[11] B. Ding, J. X. Yu, S. Wang, L. Qin, X. Zhang, and X. Lin. Finding top-k min-cost connected trees in databases. In *ICDE*, pages 836–845, 2007.

[12] 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.

[13] Y. Fang, R. Cheng, S. Luo, and J. Hu. Effective community search for large attributed graphs. *PVLDB*, 9(12):1233–1244, 2016.

[14] V. Hristidis, L. Gravano, and Y. Papakonstantinou. Efficient ir-style keyword search over relational databases. In *PVLDB*, pages 850–861, 2003.

[15] A. L. Hu and K. C. Chan. Utilizing both topological and attribute information for protein complex identification in ppi networks. *TCBB*, 10(3):780–792, 2013.

[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, H. Cheng, and J. X. Yu. Dense community detection in multi-valued attributed networks. *Information Sciences*, 314:77–99, 2015.

[18] X. Huang and L. V. Lakshmanan. Attribute truss community search. *arXiv preprint arXiv:1609.00090v3*, 2016.

[19] X. Huang, L. V. Lakshmanan, J. X. Yu, and H. Cheng. Approximate closest community search in networks. *PVLDB*, 9(4):276–287, 2015.

[20] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar. Bidirectional expansion for keyword search on graph databases. In *VLDB*, pages 505–516, 2005.

[21] M. Kargar and A. An. Discovering top-k teams of experts with/without a leader in social networks. In *CIKM*, pages 985–994, 2011.

[22] S. Khuller and B. Saha. On finding dense subgraphs. In *ICALP*, pages 597–608, 2009.

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

[24] T. Lappas, K. Liu, and E. Terzi. Finding a team of experts in social networks. In *KDD*, pages 467–476, 2009.

[25] G. Li, B. C. Ooi, J. Feng, J. Wang, and L. Zhou. Ease: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. In *SIGMOD*, pages 903–914, 2008.

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

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

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

[29] L. Qin, J. X. Yu, L. Chang, and Y. Tao. Querying communities in relational databases. In *ICDE*, pages 724–735, 2009.

[30] Y. Ruan, D. Fuhry, and S. Parthasarathy. Efficient community detection in large networks using content and links. In *WWW*, pages 1089–1098, 2013.

[31] A. E. Sariyuce, C. Seshadhri, A. Pinar, and U. V. Catalyurek. Finding the hierarchy of dense subgraphs using nucleus decompositions. In *WWW*, pages 927–937, 2015.

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

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

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

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

[36] J. Yang and J. Leskovec. Overlapping community detection at scale: a nonnegative matrix factorization approach. In *WSDM*, pages 587–596, 2013.

[37] Y. Zhou, H. Cheng, and J. X. Yu. Graph clustering based on structural/attribute similarities. *PVLDB*, 2(1):718–729, 2009.

[3] http://geneontology.org/ontology/go-basic.obo