Effective and Efficient Community Search for Complex Network Semantics Capture: From Coarse-Grain to Fine-Grain

Shuai Han
Harbin Engineering University
Harbin, China
hshuai@hrbeu.edu.cn

Yushi Tao
Harbin Engineering University
Harbin, China
yushi.tao@hotmail.com

Jingwen Tan
Harbin Engineering University
Harbin, China
ttt19@hrbeu.edu.cn

Huanran Wang*
Harbin Engineering University
Harbin, China
huanran.wang@hrbeu.edu.cn

Wu Yang*
Harbin Engineering University
Harbin, China
yangwu@hrbeu.edu.cn

Yanmei Wang
China Unicom (Heilongjiang) Company
Harbin, China
wangym36@chinaunicom.cn

ABSTRACT

To analyze the massive social networks for providing personalized services, community search is widely studied to find the densely connected subgraph that can reflect the network properties for a given query. The existing community search methods adopt single community model to make structural constraints on communities, which can only describe single interaction mode. Since they fail to capture the semantics of the network with multiple interaction modes, they struggle to find the representative communities. To solve this issue, we design a novel community model called (τ, ρ)-camp to flexibly capture complex network semantics in any level of granularity. We propose the unified support maximized community search problem to find the communities with the densest network semantics, which is proven a NP-hard problem. By constructing a hierarchical index structure, we propose an approximate community search algorithm with approximation ratio of 2 and linear time complexity of the query size. Extensive experiments are conducted on two public datasets and two crawled datasets. The experimental results prove the effectiveness and efficiency of our method.

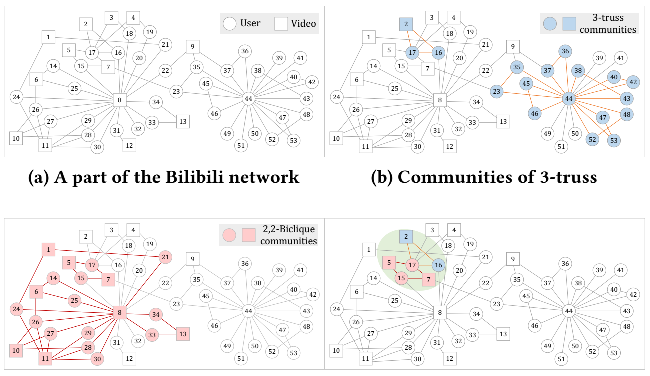
PVLDB Reference Format:
Shuai Han, Yushi Tao, Jingwen Tan, Huanran Wang*, Wu Yang*, and Yanmei Wang. Effective and Efficient Community Search for Complex Network Semantics Capture: From Coarse-Grain to Fine-Grain. PVLDB, 18(10): 3669 - 3681, 2025.
doi:10.14778/3748191.3748223

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/hsyyys/tau-rho-camp.

1 INTRODUCTION

Nowadays, social network has become an essential component in daily life [40]. To mine valuable information, the massive social network is always modeled as big graph for analysis. Instead of the big graph, it is more fruitful to focus on smaller but more cohesive subgraphs, called communities, since communities can reflect important properties of the network such as connectivity and centrality [24, 30]. To provide personalized services like advertising and recommendation [23, 37], community search is popularly studied to find a community that contains the query vertices [10, 26].

Proceedings of the VLDB Endowment, Vol. 18, No. 10 ISSN 2150-8097.
doi:10.14778/3748191.3748223



(a) A part of the Bilibili network (b) Communities of 3-truss
(c) Communities of 2,2-biclique (d) Combined community

Figure 1: Existing community models fail to capture complex network semantics.

To find such cohesive communities, researchers adopt the community models to make structural constrains on communities in traditional networks [16]. Taking two classical models for example, k-core requires that each vertex has at least k neighbors [3, 27], and k-truss restricts that each edge is contained in at least $k-2$ triangles [1, 15]. In this way, the community cohesion is guaranteed by vertex degree or stable triangle. Considering the vertex types, the networks are modeled as heterogeneous information networks (HINs). The community models are extended to HINs by building relationships between the same types of vertices based on meta-paths [11, 18, 43]. Focusing on the bipartite graphs in HINs, some community models are proposed to make constraints on bipartite graphs, such as biclique [2, 42], bitruss [25] and (α, β)-core [33].

So far, the existing community search methods adopt their own single community model to describe monotonous interaction mode to find communities [10, 16]. As a result, those methods fail to catch the complex interaction semantics of the real-world network with multiple interaction modes, which we call network semantics. For example, Figure 1(a) shows a part of the real-life social network, Bilibili, where square vertices represent videos and circle vertices represent users. If adopting the community model k-truss (k is set to 3), we obtain two communities shown in Figure 1(b). If we use the 2,2-biclique model restricting that each vertex in one type is connected to two vertices in the other type, we achieve the communities in Figure 1(c). It is obvious that these communities are scattered, as a result of depending on single interaction mode. Thus, they can not catch the complex network semantics, then can not typically reflect the overall properties of the network.

Besides, a strategy may be adopted to try to catch the complex network semantics. We can utilize two or more community models to find their respective communities and then simply combine them together. For instance, we can combine the 3-truss communities in Figure 1(b) and the 2,2-biclique communities in Figure 1(c) by the common vertices, and the result is shown in Figure 1(d). Obviously, only one 3-truss community and one 2,2-biclique community have common vertices and can be combined together, while the others are disjoint. Namely, different semantics caught by different community models are separated in this way. Hence, this strategy fails to achieve a unified and integral expression of the network semantics, thus can not reflect the network properties either.

To solve the above issue, we aim to find the communities that can reflect the network properties. There are two challenges. The first one is called effectiveness challenge. It depends on the requirement for the community model that can capture the complex network semantics to reflect the network properties. The second one is called efficiency challenge, laying on the demand for efficient search.

For the effectiveness challenge, it is challenging to come up with a community model that flexibly expresses the complex network semantics. We should first induce the different interaction modes. In the popular networks like Twitter and TikTok whose purposes are information dissemination and acquisition besides friend communication, there are both direct interaction (i.e., interaction with friends) and indirect interaction (i.e., interaction through browsing the same contents or following the same influencers) semantics. It is desirable to use the closed circulars between vertices to make restrictions on the two interaction semantics, since the closed circulars can guarantee the community cohesion. First, for direct interactions, the triangle is considered as the basic circular to produce stable restrictions [1, 4]. Then, for indirect interactions, a user has close relationship with another user if they simultaneously focus on two or more same contents or influencers, producing the basic circulars of quadrangle (e.g., two users watch two videos). Therefore, we adopt the simplest circulars, triangle and quadrangle, to design the community model to describe the communities with both direct and indirect interactions. The community model needs to catch different granularity of network semantics to understand the evolution mechanism of achieving better representative ability.

For the efficiency challenge, it is difficult to organize the massive communities with different levels of granularity into a queryable structure. Based on the community model, different communities show different granularity on the network semantics. It is essential to organize the communities from coarse-grained semantics to fine-grained semantics, so as to quickly find the communities with specific granularity of semantics for users. By exploring the nested property of the communities, we build a hierarchical index structure on the communities with different granularity of network semantics, so as to achieve efficient community search performance.

In this paper, we aim to search the communities with complex network semantics to reflect the properties of networks. We first design a novel community model named (τ, ρ)-camp to effectively capture the network semantics. (τ, ρ)-camp induces two types of interaction modes, i.e., direct interaction and indirect interaction. It describes the two semantics by the simplest circulars, triangle and quadrangle, which are also the units to form any convex polygons in the graph [7]. (τ, ρ)-camp catches different granularity of semantics by varying the restricted numbers of the two basic circulars on each edge. We propose the unified support maximized community search problem to return the communities with the densest network semantics to users, which is proven a NP-hard problem. By building a hierarchical index on communities, we can achieve search efficiency. The main contributions are summarized as below.

- To find the communities reflecting the properties of the network, we capture the complex network semantics by designing a novel community model named (τ, ρ)-camp. It organically models both direct and indirect interaction semantics in any granularity for communities.
- We propose the community search problem to find the communities with the densest complex network semantics (a NP-hard problem), or communities with users' required semantics. By leveraging the hierarchy characteristics of the built (τ, ρ)-camp index, we design a community search algorithm that adopts the greedy strategy to solve the NP-hard problem with approximation ratio of 2 in linear time.
- Extensive experiments are conducted on four datasets. The experimental results prove that our proposed algorithm is efficient in community search, and the target communities can effectively reflect the properties of the whole network.

The rest of the paper is organized as follows. The related work is discussed in Section 2. The terminologies and problem definition are introduced in Section 3. We propose the index in Section 4, and the community search method in Section 5. The experiments are conducted in Section 6. The conclusion is given in Section 7.

2 RELATED WORK

Community search aims to query a cohesive subgraph with both structure cohesiveness and high weight (significance) [33]. Existing researches based on different constraint community models are proposed to find communities on simple graph [29]. These approaches describe structural cohesion from the perspective of direct interactions [23, 35, 39]. As a well-known community model, k-core [5, 41] makes constraints on the vertex degree, i.e., the friend number of a user. Sun et al. [27] build an index called WC-index for local exploration of k-core community in the weighted graph. To find reliable communities in dynamic graph, Tang et al. [28] design (θ, k)-core model, and propose a dynamic programming based community search algorithm by constructing weighted core forest index. In terms of structural cohesion, the degree constraint of k-core is weaker than the triangle support of k-truss [34], which forms stable triangles between friends. k-truss is widely studied and extended to different variants, e.g., the triangle-connected model [1], the closest model [17] and directed D-truss [22], etc. In addition, the densest clique percolation is also applied to search communities [38], which restricts that each user is connected to any other user. Moreover, some indicators are usually introduced to these community models to further restrict the community cohesion, such as modularity [20], density [6], conductance, etc. All these community models only focus on direct interactions between users [8], and can not achieve a complete expression for the complex network semantics with both direct and indirect interactions.

Existing researches [8, 11, 43] extend the community models to heterogeneous information networks (HINs) or labeled networks. For HINs, the HIC [43] based on meta-paths is proposed to find the communities that have a dense structure and similar attributes. Similarly, Fang et al. [11] utilize the meta-path to model the cohesiveness of a community with vertices of the same type. For the labeled network, researchers aim to find two groups with different labels [2, 8, 25, 32, 33]. On the bipartite graphs, the butterfly motif (2,2-biclique) is a cohesive structure, which is often used in the bipartite graph analysis [31]. The framework of bipartite subgraphs based on the butterfly motif is defined to model the dense regions in a hierarchical structure [25]. Afterwards, the BPC-Index [2] based on bicliques is further proposed to search the biclique percolation communities, which obtains the result in near-optimal time with well-bounded index space. Based on (α, β)-core model [33], the bipartite hierarchy [32] is proposed to discover the hierarchical structure of bipartite graphs. Jiang et al. [19] combine metapaths with the k-core model to mine star-schema communities by maximizing the number of shared metapaths. However, the above methods only concern about the heterogeneous indirect interactions between different types of vertices, while ignore the direct interactions. As a result, they struggle to catch the complex network semantics and can not find communities that reflect the network properties.

The above approaches rely on a single kind of community model by considering either direct interactions or indirect interactions. The semantics captured by the existing community models are incomplete and have huge gaps with the original complex semantics of the network. Hence, they can not describe the global properties of the network, which has negative effects on the downstream tasks. To solve this problem, we propose a novel community model named (τ, ρ)-camp, which is a comprehensive expression for complex semantic network. (τ, ρ)-camp constructs more comprehensive constraint model to capture the structural characteristics of complex interaction. Moreover, we propose an efficient community search algorithm based on (τ, ρ)-camp.

3 PROBLEM DEFINITION

In this section, we introduce the relevant concepts, especially define the (τ, ρ)-camp community model and give the formal definition for the community search problem.

For generality, we extract the network to a simple undirected graph. Let $G = (V, E)$ be a graph, where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of vertices, n is the vertex number, $E = \{e_1, \ldots, e_m\}$ is the set of edges, m is the number of edges, and $\forall e_k = (v_i, v_j) \in E, v_i, v_j \in V$. $\forall v_i \in V$, $Nei(v_i) = \{v_j | (v_i, v_j) \in E\}$ is the neighbor set, and $d(v_i) = |Nei(v_i)|$ is the degree.

Definition 1. ***Community.*** *Given a graph G, C is a connected subgraph in G. If the vertices in C are densely connected to each other and sparsely connected to other vertices outside C, C is called a community in G[10].*

To design the community model to capture the complex network semantics, we first decompose the network semantics into two types from the perspective of users, i.e., the semantics of direct interaction and indirect interaction. The direct interaction refers to the friendship between users. At the meantime, users also pay attention to the indirect interaction in social networks. For example, users indirectly interact with others by browsing on the same videos, or following the same influencers. Inspired by the induction, we describe the two semantics by the simplest closed circulars, triangle and quadrangle, to form the community model. Based on the concept of triangle [30], we utilize the concept of support [16] to describe the cohesion degree of direct interaction for an edge, which we rename triangle support.

Definition 2. ***Triangle Support [16].*** *In a graph G, a triangle is a cycle of length 3. $\forall e \in E$, the triangle support of e is the number of triangles that contain e in G, denoted by $tsup(e, G)$. The support of e in a subgraph C is similarly defined and denoted by $tsup(e, C)$.*

To describe the indirect interaction semantics, we concern two perspectives. From the intuitive perspective, two users are more probabilistic to fall into one community if they browse or follow a number of same contents or influencers, while quadrangle (a circle of length 4) is the basic closed circular of such co-browsing or co-following relationships. From the theoretical perspective, we adopt the key idea of relaxation. There will be few triangles in the case of indirect interaction. So we combine two triangles and break the constraint of the overlapping edge to relax to a quadrangle to model the indirect interaction. Moreover, it is proven that triangle and quadrangle are the basic shapes to form any convex polygons by relaxation [7]. Then, we present quadrangle support to describe the cohesion degree of indirect interaction semantics for an edge.

Definition 3. ***Quadrangle.*** *For any four vertices $v_1, v_2, v_3, v_4 \in V$ in the given graph G, suppose that v_1 is the vertex with the minimal vertex id. If $(v_1, v_2) \in E$, $(v_2, v_3) \in E$, $(v_3, v_4) \in E$ and $(v_4, v_1) \in E$, we call (v_1, v_2, v_3, v_4) a quadrangle, where the vertex id of v_2 is assumed to be smaller than that of v_4.*

Definition 4. ***Quadrangle Support.*** *Given a graph G, for $\forall e \in E$, the quadrangle support of e is the number of quadrangles that contain e in G, denoted by $qsup(e, G)$. The quadrangle support of e in a subgraph C is similarly defined and denoted by $qsup(e, C)$.*

By integrating the two supports, we present the concept of unified support to represent the unified cohesion degree of a community in the aspect of complex network semantics.

Definition 5. ***Unified Support.*** *Given a community $C = (V_C, E_C)$, for $\forall e \in E_C$, the sum of the triangle support and the quadrangle support is computed, and the unified support of C is the minimum of the sums, denoted by $usup(C)$, i.e.,*

$$usup(C) = \min_{e \in E_C} \left(tsup(e, C) + qsup(e, C) \right)$$

Please note that we treat triangle support and quadrangle support equally in the above equation, since the two interaction semantics are considered of equal importance for community cohesion here. In future work, the two supports can be also allocated with different weights in this equation according to different scenarios. Based on the two supports, we give the definition of (τ, ρ)-camp community model, where camp refers to a triangle on top of a quadrangle.

Definition 6. (τ, ρ)***-camp.*** *Given a graph G, two positive integers τ and ρ, a subgraph $C = (V_C, E_C) \in G$ is called a (τ, ρ)-camp if (1) $\forall e \in E_C, tsup(e, C) \geq \tau$ or $qsup(e, C) \geq \rho$; (2) C is maximal, i.e., any supergraph $C' \supsetneq C$ is not a (τ, ρ)-camp. The granularities on the two semantics are expressed by τ and ρ.*

Next, we formally define the **Complex-Semantics based Community Search problem**, **CSCS** for short.

Definition 7. ***CSCS Problem.*** *Given a graph $G = (V, E)$, a query vertex set $Q \subseteq V$, and two positive integers τ, ρ, the CSCS problem is to find the (τ, ρ)-camp community C containing all the query vertices.*

By the expression of the CSCS problem, users can request for communities with personalized semantics by arbitrarily adjusting the values of τ and ρ. However, most of the users may be not familiar with the networks and can not provide the exact values of τ and ρ. Consequently, we will provide the users with the cohesive communities with the densest network semantics, i.e., the highest unified support. Then, we propose the **Unified-Support maximized Community Search problem**, **USCS** for short.

Definition 8. ***USCS Problem.*** *Given a graph $G = (V, E)$, a query vertex set $Q \subseteq V$ and a positive integer $\alpha \leq |V|$, the USCS problem is to find the (τ, ρ)-camp community $C_Q = (V_C, E_C)$ with the highest unified support satisfying $|V_C| \geq \alpha$ and $Q \subseteq V_C$.*

To analyze the complexity of the USCS problem, we define the decision problem corresponding to USCS, called USCSD, which we prove a NP-hard problem in the following.

Definition 9. ***USCSD Problem.*** *Given a graph $G = (V, E)$, two positive integers $\alpha \leq |V|$ and μ, the USCSD problem is to identify if there is a (τ, ρ)-camp community $C = (V_C, E_C)$ with $|V_C| \leq \alpha$ such that $usup(C) \geq \mu$.*

Theorem 1. *The USCSD problem is NP-hard.*

Proof. We will prove that the USCSD problem is NP-hard by reducing from the CLIQUE problem [14], which is a classical NP-complete problem. A CLIQUE instance can be represented by $\phi = (G_1, c)$, where $G_1 = (V_1, E_1)$, $V_1 = \{v_1, v_2, \ldots, v_{|V_1|}\}$, $E_1 = \{e_1, e_2, \ldots, e_{|E_1|}\}$, and $c \leq |V|$ is an positive integer. The CLIQUE problem is to determine if there is a clique of size c, i.e., a subset $V' \subseteq V_1$ with $|V'| = c$ such that every two vertices in V' are joined by an edge in E_1. To build an instance $I = (G, \alpha, \mu)$ of the USCSD problem, the reduction includes the following steps.

(1) Let $G = G_1$; (2) Let $\alpha = c$; (3) Let $\mu = (c-2)^2$.

Obviously, the reduction can be completed within polynomial time. Then, we need to show the correctness of the reduction. That is, the CLIQUE instance ϕ has a clique of size c if and only if the USCSD instance I has a (τ, ρ)-camp community with size not exceeding α and $usup(C) \geq \mu$.

$\Rightarrow$ If ϕ has a clique of size c, we can find a (τ, ρ)-camp community with size no larger than α such that $usup(C) \geq \mu$. For each (u, v) in the clique, we can find that it is contained in $\tau = c - 2$ triangles, and $\rho = A_{c-2}^2$ quadrangles. Hence, if ϕ has a clique of size c, then the clique is a (τ, ρ)-camp community C with size no larger than α that satisfies $usup(C) \geq \mu$.

$\Leftarrow$ If I has a (τ, ρ)-camp community C with size no larger than α and $usup(C) \geq \mu$, we can find a clique of size c. In the community, the unified support of each edge (u, v) is no less than $\mu = (c-2)^2$. Namely, besides u and v, as long as we repeatedly select twice from the other vertices, there will be a triangle or quadrangle. Particularly, if the two selected vertices are the same, a triangle is formed with u and v. There are $c - 2$ cases that the two selected vertices are the same. Namely, each of the other $c - 2$ vertices is connected to u and v. Thus, the community is a clique of size c.

Hence, we have proven the USCSD problem is NP-Hard. □

4 (τ, ρ)-CAMP INDEX

To efficiently solve the proposed community search problem, we design and construct an index for (τ, ρ)-camp communities in this section. Based on the index, we can quickly find the required (τ, ρ)-camp communities for users.

4.1 Evolution Tree

As the basis of index construction, we first present a new structure named evolution tree. It can help to order the network semantics from coarse-grain to fine-grain, so as to capture the evolution trend of the network semantics. We first define the concept of evolution for (τ, ρ)-camp community model, then build the evolution tree.

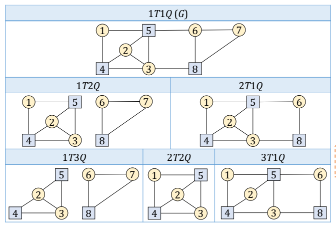
Definition 10. ***Evolution.*** *For a (τ, ρ)-camp, the evolution refers to a further strengthening of its constraints. Specifically, its possible evolution directions are either $(\tau + 1, \rho)$-camp that makes striker constraints on triangle support from the perspective of direct interaction semantics, or $(\tau, \rho + 1)$-camp with striker constraints on quadrangle support from the perspective of indirect interaction semantics.*

$$\begin{tikzcd}[column sep=tiny]
 & & & 1T1Q \arrow[dl] \arrow[dr] & & & \\
 & & 1T2Q \arrow[dl] \arrow[dr] & & 2T1Q \arrow[dl] \arrow[dr] & & \\
 & 1T3Q \arrow[dl] \arrow[dr] & & 2T2Q \arrow[dl] \arrow[dr] & & 3T1Q \arrow[dl] \arrow[dr] & \\
1T4Q & & 2T3Q & & 3T2Q & & 4T1Q \\
\cdots & & \cdots & & \cdots & & \cdots
\end{tikzcd}$$

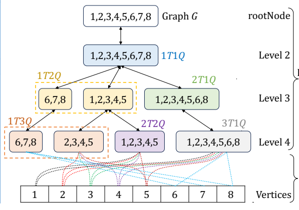
Figure 2: The evolution tree.

As shown in Figure 2, the most relaxed version of (τ, ρ)-camp is $(1, 1)$-camp, denoted as $1T1Q$ for short, where 'T' refers to triangle support and 'Q' represents quadrangle support. By making striker restrictions for $1T1Q$, it evolves into $2T1Q$ or $1T2Q$. Similarly, $2T1Q$ can evolve into $3T1Q$ and $2T2Q$, and $1T2Q$ grows into $2T2Q$ and $1T3Q$. We repeatedly execute the above steps and generate the evolution tree shown in Figure 2. As the level increases, the network semantics gets more fine-grained on the aspect of either direct or indirect interaction semantics. We can achieve any granularity combination of the two semantics by using the evolution tree.

Following the evolution tree, the (τ, ρ)-camp community is proven to have the nested property in Theorem 2. Based on this property,



(a) The (τ, ρ)-camp communities



(b) The constructed index

Figure 3: An illustration of the (τ, ρ)-camp index.

we can get the (τ, ρ)-camp communities by recursive decomposition and build index on them for efficient community search.

Theorem 2. *Given a (τ, ρ)-camp community $C = (V_C, E_C)$, where $\tau > 0, \rho > 0$, let $C' = (V_{C'}, E_{C'})$ be a $(\tau + 1, \rho)$-camp or $(\tau, \rho + 1)$-camp community evolving from C. Then, we have $V_{C'} \subseteq V_C$ and $E_{C'} \subseteq E_C$, i.e., C' is nested within C.*

proof sketch. By enumerating the cases of $(\tau+1, \rho)$-camp and $(\tau, \rho+1)$-camp, it is proven $E_{C'} \subseteq E_C$ and $V_{C'} \subseteq V_C$ by using stricter constraints on either triangle support or quadrangle support. □

Theorem 3. *The diameter of a (τ, ρ)-camp community C with m_C edges is bounded by $diameter_C \leq \max\{\frac{2(m_C-1)}{\tau}, \frac{2(m_C-1)}{\rho+2}\}$.*

proof sketch. Similar to the proof of the diameter upper bound on k-truss [4], we compute the diameter upper bound for a (τ, ∞)-camp, $\frac{2(m_C-1)}{\tau}$, and for a (∞, ρ)-camp community, $\frac{2(m_C-1)}{\rho+2}$. □

4.2 Index Model

This subsection introduces the index model for (τ, ρ)-camp communities, called *TQIndex*, for efficient community search.

Following the evolution tree, the given graph G can be decomposed into nested (τ, ρ)-camp communities with different values of τ and ρ, based on which *TQIndex* is built. Specifically, *TQIndex* consists of two parts, i.e., an index tree and a map, denoted as:

$TQIndex = (IndexTree, Map, Maxlevel)$

$IndexTree = (root, TQCom, R)$

$Map = \{\langle v, CSet\rangle \mid \forall C \in CSet, C \text{ is a leaf containing } v\}$.

In *TQIndex*, *IndexTree* is an index tree built on multiple (τ, ρ)-camp communities. Specifically, *root* is the root node of *IndexTree*, i.e., the graph G. The other nodes in *IndexTree* represent the (τ, ρ)-camp communities, storing in the community set *TQCom*. Following the evolution tree, the communities can be decomposed into smaller nested communities with stricter restrictions, i.e., bigger value of τ or ρ. R stores the nested relationships between the communities. When the communities reach the maximal level *Maxlevel* or can not be decomposed further, they are the leaves of *IndexTree*. Obviously, for a (τ, ρ)-camp community, its level is $\tau + \rho$. Each vertex is mapped to the leaf communities that contain it, and *Map* stores the mappings. Next, we give an illustration of *TQIndex*.

Example 1. *As shown in Figure 3(a), given a graph G with vertices $\{1, 2, 3, 4, 5, 6, 7, 8\}$, it is a native (1,1)-camp community. Then, it can be decomposed into two (1,2)-camp communities $\{1, 2, 3, 4, 5\}$ and $\{6, 7, 8\}$, or decomposed into one (2,1)-camp community $\{1, 2, 3, 4, 5, 6, 8\}$. Furthermore, the (1,2)-camp community $\{6, 7, 8\}$ is also (1,3)-camp. The (1,2)-camp community $\{1, 2, 3, 4, 5\}$ is also (2,2)-camp, and can evolve into (1,3)-camp community $\{2, 3, 4, 5\}$. Let $Maxlevel = 4$. The nested relationships form the index tree in Figure 3(b), and the vertices are mapped to the leaf communities that contain them. E.g., the vertex 6 is mapped to (1,3)-camp $\{6, 7, 8\}$ and (3,1)-camp $\{1, 2, 3, 4, 5, 6, 8\}$.*

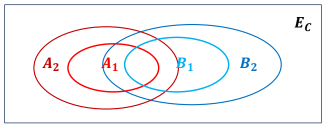


Figure 4: An illustration for the proof of Lemma 1.

Lemma 1. *Given a (τ, ρ)-camp community $C = (V_C, E_C)$, there are two paths to generate the $(\tau+1, \rho+1)$-camp communities along the evolution tree, i.e., $path_1$: $\tau T\rho Q \rightarrow (\tau+1)T\rho Q \rightarrow (\tau+1)T(\rho+1)Q$, or $path_2$: $\tau T\rho Q \rightarrow \tau T(\rho+1)Q \rightarrow (\tau+1)T(\rho+1)Q$. The generated $(\tau+1, \rho+1)$-camp communities along the two paths are the same.*

Proof. Let A_1 be the set of edges whose triangle supports are τ in C. When first making stricter constraints on triangle support (i.e., $(\tau+1)T\rho Q$), A_1 should be deleted. Due to the deletion of A_1, some edges' supports change and may also be deleted. Similarly, some other edges continue to be deleted due to the deletion of the affected edges. This procedure continues until there are no more edges to be deleted. The set of all the deleted edges is denoted as A_2. Obviously, $A_1 \subseteq A_2$. B_1 and B_2 are similarly defined when first making stricter constraints on quadrangle support (i.e., $\tau T(\rho+1)Q$). As shown in Figure 4, A_1 and A_2 may intersect with B_1 and B_2.

Along $path_1$, we first delete A_2 from the edge set E_C of the community C at the step of $(\tau+1)T\rho Q$. When deleting A_2, a subset of B_2 is deleted, i.e., $A_2 \cap B_2$. At the second step of $(\tau+1)T(\rho+1)Q$, we delete $B_2 - A_2$ from the remained edge set $E - A_2$, so we finally get the edge set $E - A_2 \cup B_2$ for the $(\tau+1, \rho+1)$-camp community along $path_1$. Similarly, it is proven that we also get $E - B_2 \cup A_2$ for the $(\tau+1, \rho+1)$-camp along $path_2$. Hence, Lemma 1 is proven. □

Based on Lemma 1, we can obtain the following theorem.

Theorem 4. *Based on the evolution tree, the given graph G is inputted and follows several paths, e.g.,*
$path_1$: $1T1Q \rightarrow \cdots \rightarrow (\tau-1)T\rho Q \rightarrow \tau T\rho Q$,
$path_2$: $1T1Q \rightarrow \cdots \rightarrow \tau T(\rho-1)Q \rightarrow \tau T\rho Q$, ...,
to get the (τ, ρ)-camp communities. Then, for all the possible paths, the obtained communities are the same.

Proof. In each step, if making stricter restrictions on triangle support, 'T' is used to represent this step, otherwise 'Q' is used. In this way, each path is translated to a sequence of 'T's and 'Q's whose numbers are τ and ρ, respectively. In Lemma 1, we have proved that $TQ = QT$ if starting from a same community, so $TQTQ = TTQQ$.

For each path, i.e., a sequence composed of 'T's and 'Q's, we find the position of QT, and exchange them. This operation is repeated until there is no QT. At this time, we transform the sequence to,

$$\underbrace{T \cdots T}_{\tau} \underbrace{Q \cdots Q}_{\rho}$$

Hence, the obtained (τ, ρ)-camp communities along all the possible paths are the same. □

4.3 Index Construction

After introducing the (τ, ρ)-camp index model, we propose the index construction algorithm. Based on the nested property, the algorithm obtains the nested communities following the evolution tree. Specifically, this algorithm mainly contains three steps. In the first step, it computes the triangle support and quadrangle support for each edge. In the second step, it decomposes the graph along the evolution tree to achieve all the (τ, ρ)-camp communities no exceeding the maximal level. In the third step, it builds the index upon the communities. Next, we introduce the algorithm in details.

4.3.1 Algorithm description. The formal description of the index construction algorithm is shown in Algorithm 1. The input is the graph G and the maximal level of the index tree $Maxlevel$, and the output is the built index $TQIndex$. The root of the index tree is set as the biggest community, i.e., the graph G (*line 1*). In the first step of computing the triangle support and quadrangle support, we find the neighbors for each vertex (*line 2-3*). For each edge $e = (u, v) \in E$, its triangle support is the size of the intersection of $Nei(u)$ and $Nei(v)$ (*line 5*). To count the quadrangle support, we enumerate each neighbor of u, denoted as v', and accumulate the size of the intersection of $Nei(v)$ and $Nei(v')$ (*line 6-8*).

In the second step, we first push the biggest community, G, into the queue of communities to be decomposed (*line 9*). When the queue is not empty, we pop a (τ, ρ)-camp community $C = (V_C, E_C)$ from the queue for decomposition along two evolution directions explained as below. When improving triangle support, we find the edges that do not satisfy the constraints of $(\tau+1, \rho)$-camp. We delete these edges, and the triangles and quadrangles containing the edges will disappear. Then, the triangle/quadrangle supports of other edges in those triangles/quadrangles will decrease. As such, some of those affected edges should also be deleted. The above procedure is repeatedly executed until there is no edge to delete (*line 14-19*). When improving quadrangle support, we similarly delete the edges whose triangle supports are less than τ and quadrangle supports are less than $\rho+1$ (*line 20-25*). The sets of remained edge for the two cases are denoted as E_1, E_2. The connected algorithm is processed on E_1 and E_2 to get the new $(\tau + 1, \rho)$-camp and $(\tau, \rho + 1)$-camp communities, each of which is pushed into the queue if lower than $Maxlevel$ (*line 26*). The loop is executed until the queue is empty.

In the third step, the communities and their relationships form the index tree, and the vertices are mapped to their leaf communities (*line 27*). Finally, the built index $TQIndex$ is outputted (*line 28*).

Please notice that for a (τ, ρ)-camp community with $\tau > 1$ and $\rho > 1$, it can be either evolved from its parent $(\tau - 1, \rho)$-camp community or $(\tau, \rho - 1)$-camp community. Hence, we only need to generate this (τ, ρ)-camp community from either of its two parents. We store the community ids of the community nodes in the index tree in memory for efficient search. The edges of the communities are stored on disk. Next, we illustrate the index construction steps.

Algorithm 1: (τ, ρ)-camp index construction algorithm

```
Input: The graph $G = (V, E)$, the maximal level $Maxlevel$.
Output: The index $TQIndex = (IndexTree, Map)$.
1  $IndexTree.root \leftarrow G$
   /* Step 1. compute the two supports */
2  for $v_i \in V$ do
3      $Nei(v_i) \leftarrow \{v_j | (v_i, v_j) \in E \ or \ (v_j, v_i) \in E\}$
4  for $e = (u, v) \in E$ do
5      $tsup(e, G) \leftarrow |Nei(u) \cap Nei(v)|$
6      $qsup(e, G) \leftarrow 0$
7      for $v' \in Nei(u)$ do
8          $qsup(e, G) \leftarrow qsup(e, G) + |Nei(v) \cap Nei(v')| - 1$
   /* Step 2. generate $(\tau, \rho)$-camp communities */
9  $queue.push(G)$
10 while $queue \neq \emptyset$ do
11     $C = (V_C, E_C) \leftarrow queue.pop()$
12     $\tau \leftarrow C.trusslevel, \rho \leftarrow C.quadlevel$
13     $E_1 \leftarrow E_C, E_2 \leftarrow E_C$
14     while $\exists e \in E_1, tsup(e) < \tau + 1$ and $qsup(e) < \rho$ do
15         remove $e$ from $E_1$
16         for each edge in a same triangle with e do
17             $tsup(e', E_1) \leftarrow tsup(e', E_1) - 1$
18         for each edge in a same quadrangle with e do
19             $qsup(e', E_1) \leftarrow qsup(e', E_1) - 1$
20     while $\exists e \in E_2, qsup(e) < \rho + 1$ and $tsup(e) < \tau$ do
21         remove $e$ from $E_2$
22         for each edge in a same triangle with e do
23             $tsup(e', E_2) \leftarrow tsup(e', E_2) - 1$
24         for each edge in a same quadrangle with e do
25             $qsup(e', E_2) \leftarrow qsup(e', E_2) - 1$
26     get connected subgraphs from $E_1, E_2$, and push the
           subgraphs lower than $Maxlevel$ into $queue$
   /* Step 3. build $(\tau, \rho)$-camp index */
27 build $IndexTree$ and $Map$
28 return $TQIndex$
```

Example 2. *To continue with Example 1, the graph G contains vertices $\{1, 2, 3, 4, 5, 6, 7, 8\}$ as shown on the top of Figure 3(a). Along the evolution tree, we first delete the edges in G whose triangle supports are less than 1 and quadrangle supports are less than 1, and obtain the (1,1)-camp community which is also G. To obtain (1,2)-camp communities, the edges whose quadrangle supports are less than 2 and triangle supports are less than 1 are recursively deleted. The deleted edges are (5,6) and (3,8), and two subgraphs are built by the remained edges, which are (1,2)-camp communities {1,2,3,4,5} and {6,7,8}. Also, to obtain (2,1)-camp communities, the edges whose triangle supports are less than 2 and quadrangle supports are less than 1 are deleted, which are (6,7) and (7,8). We can get the (2,1)-camp community {1,2,3,4,5,6,8}. It continues until reaching the maximal level. Finally, we can build the index tree and map the vertices to their leaf communities in Figure 3(b).*

By using the built index, $TQIndex$, we can figure out the target communities to satisfy users' personalized requirements for communities with specific network semantics by inputting the query vertices and the values of τ, ρ, helping to solve the CSCS problem.

4.3.2 Complexity analysis. We analyze the time complexity and space complexity of the index construction algorithm.

Theorem 5. *The time complexity of the (τ, ρ)-camp index construction algorithm is $O(m)$, where m is the number of edges in G.*

Proof. We first analyze the time cost of the first step in Algorithm 1, which mainly depends on the enumeration of triangles and quadrangles. To compute the triangle support for each edge $e = (u, v) \in E$, it costs $\min\{d(u), d(v)\}$ time to enumerate the triangles containing e. To compute the quadrangle support for e, it costs $\sum_{v' \in Nei(u)} \min\{d(v), d(v')\}$ time to enumerate the quadrangles containing e. To sum up, the time cost of the first step is,

$$\sum_{(u,v)\in E} \big(\min\{d(u), d(v)\} + \sum_{v' \in Nei(u)} \min\{d(v), d(v')\} \big)$$

where $d(u)$ is the degree of u. The extremum of the vertex degree is a constant [13], thus the first step spends $O(m)$ time.

In the second step, there are k times of decomposition to generate the k-th level. Then, the deleted edges, triangles and quadrangles will be deleted at most k times, where $k \leq l$ and l is the extremum of $Maxlevel$, i.e., the maximum of the possible levels in $IndexTree$. l is smaller than the summation of the maximal triangle support and the maximal quadrangle support. So, l depends on the degree and is a constant. Since enumerating all the triangles and quadrangles costs $O(m)$ time, the second step spends $O(lm) = O(m)$ time.

Then, we analyze the third step. There are $2(1+2+3+\cdots+l-1) = O(l^2)$ branches in the evolution tree. Please note that the diameter of a (τ, ρ)-camp C is $O(\frac{m_C}{k})$, where $k = \min\{\tau, \rho\}$, and m_C is the number of edges in C. Thus, if a (τ, ρ)-camp community evolves into several $(\tau + 1, \rho)$-camp or $(\tau, \rho + 1)$-camp communities, the number of the obtained connected communities can be considered as a constant. Hence, there are totally $O(l^2)$ relationships between communities in the index tree. Each vertex is mapped to at most l leaf communities, so the mapping costs $O(ln) = O(n)$ time, where $n \leq m + 1$ is the number of vertices in the connected graph G.

In summary, the time complexity of the index construction algorithm is $O(m)$, where m is the number of edges in G. □

Theorem 6. *The space complexity of the (τ, ρ)-camp index construction algorithm is $O(m)$, where m is the edge number of G.*

Proof. The space cost of the first step mainly lies on the storing of the neighbors. Since the degree is a constant, the first step occupies $O(n)$ space, where n is the vertex number in G. In the second step, the enumeration of all the triangles and quadrangles is $O(m)$. The space cost of the queue is $O(lm)$. Then, the second steps occupies $O(lm) = O(m)$ space. In the third step, the number of all the (τ, ρ)-camp communities in the index tree is $1+2+3+\cdots+l = O(l^2)$, so storing their ids costs $O(l^2)$ space. Because there are totally $2(2+3+\cdots+l-1) = O(l^2)$ relationships between communities, it costs $O(l^2)$ space to store the relationships. Since each vertex is mapped to at most l leaf communities, it costs $O(ln)$ space to store the mappings. Thus, the third step occupies $O(ln + l^2) = O(n)$ space.

Hence, the index construction algorithm costs $O(m)$ space. □

5 COMMUNITY SEARCH METHOD

In this section, we present the community search algorithm to return the target communities to users. We first solve the basic CSCS problem. Then, we optimize the index structure, and present the **Unified-Support maximized Community Search Algorithm**, **USCSA** for short, to solve the USCS problem.

5.1 Basic Community Search

Based on the index $TQIndex$, we can find the target communities to satisfy users' personalized requirements for communities with specific network semantics, so as to solve the CSCS problem. Given the values of τ, ρ and a query q, we find the ancestor (τ, ρ)-camp community C'_v for the mapped communities of the query vertices. The target community should be the intersection of all the $\{C'_v\}$.

It can be seen that if all the C'_v corresponding to the query vertices are the same, this community is the target community. Otherwise, the target community is empty, and can not satisfy user's personalized requirement. Moreover, most users are not capable to know the exact values of τ and ρ since they do not have the background knowledge. Thus, we present the USCS problem which is defined in Section 3. It aims to find the most cohesive community with maximized unified network semantics for the given query, while the user do not need to provide the values of τ and ρ in prior.

5.2 Optimized Community Search

To return the communities that can reflect the network properties to users, we find the communities with the maximized unified network semantics. We first introduce the basic idea to solve the USCS problem, then propose the community search algorithm.

5.2.1 Basic idea. In order to efficiently solve the NP-hard USCS problem, we intend to leverage the hierarchy characteristics of the (τ, ρ)-camp index, and design an approximation algorithm to solve the USCS problem in linear time.

We adopt the key idea of greedy strategy to design the approximation algorithm. For each current (τ, ρ)-camp community, we can generate new communities along two evolution directions, which have different unified supports. To address the USCS problem, we only want to find the communities with the densest network semantics, i.e., maximized unified support, for the query vertices. Hence, we adopt the greedy strategy that compares the unified supports of the communities along the two directions, and choose the direction with higher unified support in each step. As such, we can always achieve the most cohesive communities in the current step. The generated communities in different levels form the optimized evolution paths, which compose of a subtree in the index tree. In the subtree, the communities are regarded to have the approximately maximized unified supports for their vertices in the current levels.

5.2.2 Community search algorithm. Based on the basic idea, we propose the optimized index construction algorithm to get the optimized index. The input of the algorithm is the graph G. The output is the optimized index denoted as $optIndex$, and it includes two parts, i.e., the optimized subtree denoted by $optTree$, and the map denoted by $optMap$, which stores the mappings from the vertices to their leaf communities in $optTree$.

$optIndex = (optTree, optMap)$

$optTree = (root, optCom, optR)$

$optMap = \{\langle v, optC\rangle \mid v \in optC, \forall C' \in \{C_i | v \in C_i, C_i \in optCom\}, optC.level \geq C'.level\}$.

Algorithm 2: Optimized index construction algorithm

```
Input: The graph $G = (V, E)$.
Output: The optimized index $optIndex$.
1  $optTree.root \leftarrow G$
   /* Step 1. compute the two supports */
2  compute the neighbor set $Nei(v)$ for each node $v \in V$
3  compute $tsup(e)$ and $qsup(e)$ for each edge $e \in E$
   /* Step 2. get optimized $(\tau, \rho)$-camps */
4  $queue.push(G)$
5  while $queue \neq \emptyset$ do
6      $C = (V_C, E_C) \leftarrow queue.pop()$
7      $\tau \leftarrow C.trusslevel, \rho \leftarrow C.quadlevel$
8      $E_1 \leftarrow E_C, E_2 \leftarrow E_C$
9      while $\exists e \in E_1, tsup(e) < \tau + 1$ and $qsup(e) < \rho$ do
10         remove $e$ from $E_1$
11         update the two supports for the affected edges
12     while $\exists e \in E_2, qsup(e) < \rho + 1$ and $tsup(e) < \tau$ do
13         remove $e$ from $E_2$
14         update the two supports for the affected edges
15     if $usup(E_1) \geq usup(E_2)$ then
16         get connected subgraphs from $E_1$, push into $queue$
17     else
18         get connected subgraphs from $E_2$, push into $queue$
   /* Step 3. build $(\tau, \rho)$-camp index */
19 build $optTree$ and $optMap$
20 return $optIndex$
```

The description of the optimized index construction algorithm is shown in Algorithm 2. Similarly, the algorithm includes three steps. The root of $optTree$ is set as the graph G (*line 1*). The first step computes the triangle supports and quadrangle supports for the edges in G (*line 2-3*), which is the same with Algorithm 1. The second step is to generate the optimized communities. The graph G is first put into the queue of communities to be decomposed (*line 4*). When decomposing a (τ, ρ)-camp community popped from the queue, we generate both the $(\tau, \rho+1)$-camp communities and $(\tau+1, \rho)$-camp communities in two evolution directions (*line 6-14*). We determine the better evolution direction by comparing their unified supports, then only store and put the communities of the better direction into the queue (*line 15-18*). The loop from line 5 to line 18 is continuously executed till the queue is empty. In the third step, we build $optTree$, and map each vertex to its highest community by $optMap$(*line 19*). Finally, the optimized index $optIndex$ is returned (*line 20*).

Based on the constructed optimized index, we propose the community search algorithm to return the target communities with the highest cohesion degree to users. The **U̲nified-S̲upport maximized C̲ommunity S̲earch A̲lgorithm**, called **USCSA** for short, is described in Algorithm 3.

Algorithm 3 is an approximation algorithm to solve the NP-hard USCS problem. Its input is the optimized index $optIndex$ and the query set Q, and the output is the target community with the approximate maximized unified support. For each query vertex v, the algorithm first finds its mapped optimized community C_v according to the optimized map (*line 1-2*). Then, the target community $optC$ is the highest common ancestor community of all the C_v in the optimized tree (*line 3*), and is returned to the user (*line 4*).

Algorithm 3: Unified-support maximized community search algorithm, *USCSA*

```
Input: The optimized index $optIndex$, the query set $Q$.
Output: The target community $C$.
1  for $v \in Q$ do
2      find its mapped optimized community $C_v$ by $optMap$
3  $optC \leftarrow$ the highest common ancestor community of the
       communities in $\{C_v | v \in Q\}$
4  return $optC$
```

5.2.3 Algorithm analysis. In this part, we analyze the approximation ratio and time complexity of the *USCSA* algorithm.

Theorem 7. *optC is an approximate solution for the USCS problem with the approximation ratio of 2.*

Proof. For a given query set Q, let $A = optC$ be the optimal approximate solution, which is a (τ^A, ρ^A)-camp community with the unified support of μ_A. Let D^* be the optimal solution, which is a (τ^*, ρ^*)-camp community with the unified support of μ^*. Let D be the highest ancestor optimized community of A such that $D^* \subseteq D$, called a feasible approximate solution. It is assumed that D is a (τ^D, ρ^D)-camp community with the unified support of μ^D.

First, because D is an ancestor of A, i.e., $A \subseteq D$, and

$$\mu_D = \min_{e \in D}(tsup(e) + qsup(e)),\ \mu_A = \min_{e \in A}(tsup(e) + qsup(e)),$$

we have $\mu_D \leq \mu_A$.

Then, let C be the child optimized community of D in the next level. C is either a (τ^D+1, ρ^D)-camp or a (τ^D, ρ^D+1)-camp community. We first assume C is a (τ^D+1, ρ^D)-camp community. Because $D^* \subseteq D$, the edges in D^* satisfy either triangle support no less than τ^D or quadrangle support no less than ρ^D. Because $D^* \nsubseteq C$, then $\exists e \in D^*$ such that $tsup(e) = \tau^D$ and $qsup(e) < \rho^D$. Therefore, we have

$$\mu^* = \min_{e \in D^*}(tsup(e) + qsup(e)) < \tau^D + \rho^D.$$

It is also proven for the case that C is a (τ^D, ρ^D+1)-camp community. In terms of the feasible approximate solution D, it is obvious that $\tau^D \leq \mu_D$ and $\rho^D \leq \mu_D$. Then, we have

$$\mu^* < \tau^D + \rho^D \leq 2\mu_D \leq 2\mu_A, \text{ and } \frac{\mu^*}{\mu_A} \leq 2.$$

Hence, we get the approximation ratio $\gamma = 2$. □

5.2.4 Time complexity. If there is only one query vertex, the time complexity of the *USCSA* algorithm is $O(1)$ by directly finding the mapped community of the query vertex. Otherwise, we should further find the highest common ancestor community for the mapped communities of the query vertices in $optIndex$. Recall that the number of levels in the index tree is a constant. The time complexity of the *USCSA* algorithm is $O(|Q|)$, linear of the size of the query set.

6 EXPERIMENTS

6.1 Experimental Setup

Datasets. The experiments adopt four datasets. DBLP [18] and Youtube [36] are public, while Weibo and Bilibili are collected by our crawler and desensitized. The details of the datasets are shown in Table 1. There are both direct and indirect interactions in the four datasets. In DBLP, there are direct interactions of colleague between authors, and indirect interaction through cooperating papers. In Youtube, besides the direct friendships, users also have indirect interaction through focusing on influencers. With respect to Bilibili and Weibo, users have indirect interaction through browsing videos or following influencers besides direct friend relationships.

Table 1: The dataset description.

Dataset	HIN	Users	Contents	U-U	U-C	d_u	d_c
DBLP	Yes	785,204	1,141,266	283,298	3,042,798	754	114
Youtube	No	513,278	0	2,987,624	0	28,754	0
Bilibili	Yes	4,854,514	26,531	1,058,376	8,679,490	3,370	9,796
Weibo	Yes	1,142,478	1,936,763	163,620	4,023,313	4,378	49,473

Competitors. To compare with *USCSA*, four state-of-the-art community search methods are adopted, whose details are as below.

• *CSRTI* [34]: It designs a k-truss index and introduces a relaxed level of communities obtained by community detection methods.

• *WC-index* [27]: As a traditional search method, it builds an index through a local exploration of k-core community.

• *CSSH* [18]: It is a meta-path based community search method over large star-schema heterogeneous networks.

• *RECEIPT* [21]: It finds the butterfly motif (i.e., 2,2-bicliques) based communities with restrictions on edges in bipartite graphs, and achieves a high degree of parallelism to speed up searching.

• *TIP* [25]: It is a peeling algorithm to find communities by using butterfly motif to measure vertex participations in bipartite graphs.

• *HCBU* [32]: It discovers the bipartite hierarchy and build index for efficient community search based on (α, β)-core.

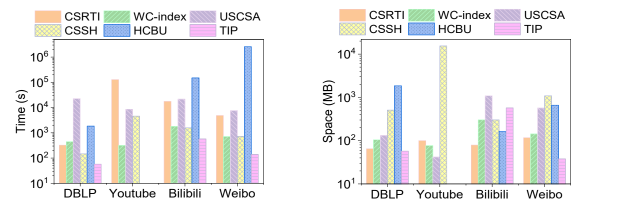
In *WC-index*, we set the edge weight as 1. In *CSSH*, we use its default parameters, e.g., the maximal length of meta-paths is 4. Please notice that the public code of *CSSH* consumes much time and memory, so we rewrite its code.

Queries. To evaluate the community search performance on each dataset, by varying the query size (i.e., the query vertex number) of 1, 2, 3, 4 and 5, we randomly generate 100 queries, respectively. By varying the data size from 20%, 40%, 60%, 80% and 100% of the datasets, we also randomly generate 100 queries, respectively.

Hardware setting. The experiments are run on two servers with an Intel Xeon Gold 5320 processor of 2.20GHz. The processor has 52 cores (26 cores per socket, 2 sockets) and a 78 MiB L3 cache. The server has a 1.7TB disk and 768GB DDR4 DRAM. The OS is 64-bit Ubuntu 22.04.4. The programs are compiled using g++ 11.4.0.

6.2 Index Construction Performance

In this experiment, we compare the preprocessing performance on index construction of *USCSA* and the baselines. On the datasets, the index construction time costs and index sizes are shown in



(a) Time cost **(b) Space cost**

Figure 5: The preprocessing performance on index building.

Fig 5(a) and (b), respectively. Please notice that *RECEIPT* does not construct index before searching, so we do not compare it here.

As shown in Figure 5(a), the index building time costs of *USCSA* and *CSRTI* are higher than other methods. Taking Youtube dataset as an example, *CSRTI* spends 1.29×10^5s, which is 15.02, 28.23, and 40.57 times of that of *USCSA*, *CSSH* and *WC-index*. *WC-index* has short index building time since it only counts the neighbors. *CSRTI* needs to generate the basic communities by community detection, which consumes expensive time costs. *USCSA* builds hierarchical index on the (τ, ρ)-camp communities, so it costs long time.

Then, Figure 5(b) shows the occupied spaces of the built indexes of the community search methods. It is found that the index of *CSSH* is the largest in most cases. For example, *CSSH* occupies 1088 MB for index on Weibo dataset, which is 9.30, 7.61 1.91, 2.30 and 18.03 times of *CSRTI*, *WC-index*, *USCSA*, *HCBU* and *TIP*. *CSSH* needs to store many multi-hop neighbors along the meta-paths, then its relationship explosion spends expensive space costs.

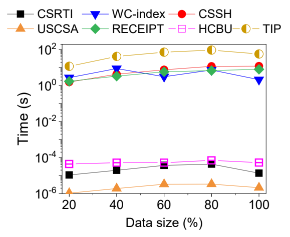
Although sacrificing the preprocessing time for building index and the space for index storage, *USCSA* will achieve efficient community search, which will be proved in the later experiment.

6.3 Community Search Efficiency

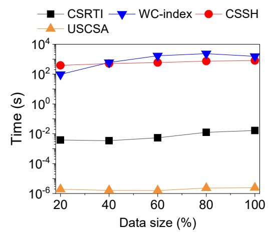
In this subsection, we evaluate the community search efficiency on different datasets. By varying the data size and the query size, i.e. the number of vertices in the query, we evaluate their effects on search efficiency for different methods in this experiment.

6.3.1 Varying the data size. We randomly select 20%, 40%, 60%, 80% data from each dataset. The query size is set as 1 in default. By varying the data size from 20% to 100%, the average community search time on each dataset is shown from Figure 6(a) to (d).

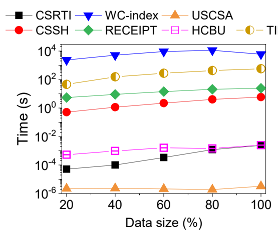
It is seen that *USCSA* performs fastest and *CSRTI* is the second fastest, while *CSSH* and *WC-index* are quite slow. Taking 60% Weibo dataset for example, *WC-index* spends 4.15×10^2s, 3.82 times of *TIP*, 5.81 times of *CSSH*, 62.97 times of *RECEIPT*, six orders of magnitude slower than *CSRTI* and *HCBU*, and eight orders of magnitude slower than *USCSA*. By loading the built index into memory, *USCSA*, *CSRTI* and *HCBU* achieve efficient searching. Since *USCSA* only needs to find the mapped community for the query vertex, *USCSA* performs faster than *CSRTI* and *HCBU*. *WC-index* has to repeatedly invoke the connected component algorithm for each deleted vertex. *CSSH* searches the neighbors of the query vertex based on meta-path layer by layer, which are too massive to traverse. *RECEIPT* and *TIP* are slower than *USCSA* and *CSRTI* since *RECEIPT* builds no index and *TIP* builds light index. *RECEIPT*



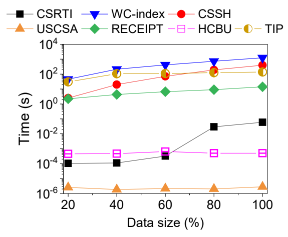
(a) DBLP, data size



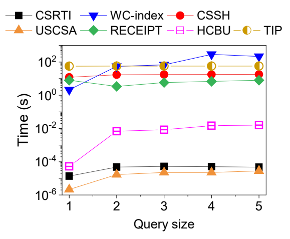
(b) Youtube, data size



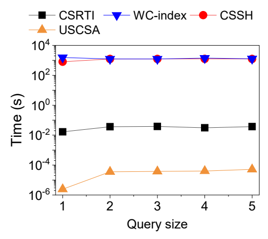
(c) Bilibili, data size



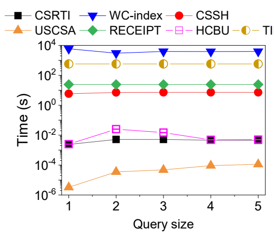
(d) Weibo, data size



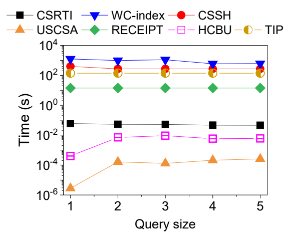
(e) DBLP, query size



(f) Youtube, query size



(g) Bilibili, query size



(h) Weibo, query size

Figure 6: The experimental results of community search time costs.

achieves a high degree of parallelism and dramatic reduction in synchronizations, so it performs better than *WC-index* and *CSSH*, especially on the larger datasets. *RECEIPT*, *HCBU* and *TIP* can not process Youtube with only one vertex type.

Please note that many communities found by *CSSH* for different queries are the same, since the explosive relationships generated by meta-paths break the boundaries of normal communities.

6.3.2 *Varying the query size.*

We randomly generate 100 queries for the query set size of 1, 2, 3, 4 and 5 on different datasets, respectively. We use 100% data in default. By varying the query size, the community search time results are shown from Figure 6(e) to (h).

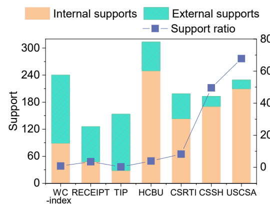
On four datasets, *USCSA* obtains the shortest community search time for different query sizes. For example, when processing queries in the size of 2 on Youtube dataset, *WC-index* spends the longest time of 1.24×10^3s and *CSSH* costs 1.23×10^3s, while *CSRTI* and *USCSA* spends 3.66×10^{-2}s and 3.57×10^{-5}s, respectively. It should be noted that the meta-path based *CSSH* spends not only long search time but also a great deal of memory, especially for queries with more than one vertices, since its enumeration space is massive.

As the query size increases from 1 to 2, the time cost of *USCSA* increases shapely. The reason is analyzed as follows. When there is only one query vertex, *USCSA* just needs to return the mapped community of the vertex, which is $O(1)$ time. For more vertices, *USCSA* searches upwards for the highest common ancestor community of the mapped communities, linear time of the query size.

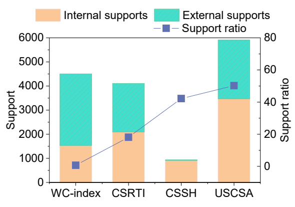
Proved by the two experiments, *CSRTI* achieves efficient community search performance based on the built index. Therefore, it deserves to consume the preprocessing time and space for index.

6.4 Community Search Effectiveness

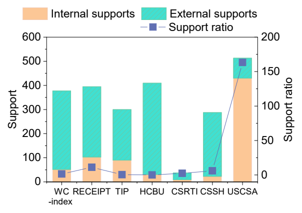
This experiment is to prove the effectiveness of our proposed *USCSA* algorithm. Since the four datasets have no answers of real communities, which is common in real-world scenarios, we do



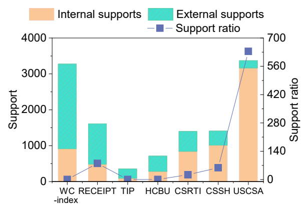
(a) DBLP, support ratio



(b) Youtube, support ratio



(c) Bilibili, support ratio



(d) Weibo, support ratio

Figure 7: The experimental results of support ratio.

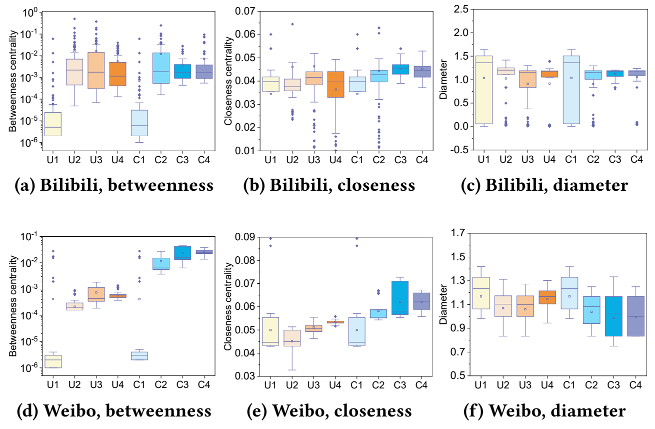
not adopt the metrics such as precision and recall to evaluate the effectiveness of community search. Instead, we first measure the cohesion degree of communities, and then evaluate the representative ability, which is the motivation for the proposal of community.

6.4.1 *Cohesion evaluation.*

We adopt four metrics to measure the community cohesion. The first metric is support ratio, i.e., the average of ratio between the internal support and the external support of each edge, to evaluate community compactness in terms of the network semantics. The second metric is degree ratio that measures the degree comparison between inside and outside of the community [12]. Higher degree ratio means the community fully covers the vertices and their neighbors within the community, reserving

Table 2: The experimental results of degree ratio, local modularity and conductance.

Dataset	Degree ratio							Local Modularity							Conductance						
	WC	REC	TIP	HCBU	CSRTI	CSSH	USCSA	WC	REC	TIP	HCBU	CSRTI	CSSH	USCSA	WC	REC	TIP	HCBU	CSRTI	CSSH	USCSA
DBLP	1.09	1.51	1.86	1.89	3.79	2.71	**4.65**	0.067	0.044	0.113	0.037	0.215	0.022	**0.277**	0.892	0.875	0.773	0.944	0.673	0.958	**0.517**
Youtube	0.72	——	——	——	**5.04**	2.58	4.68	0.107	——	——	——	0.265	**0.761**	0.157	0.810	——	——	——	0.607	**0.136**	0.537
Bilibili	1.82	2.28	2.14	3.70	1.73	1.53	**4.95**	0.116	0.171	0.246	0.131	0.305	0.254	**0.384**	0.805	0.646	0.529	0.784	0.570	0.609	**0.421**
Weibo	0.54	0.75	2.45	**3.37**	2.53	2.69	1.31	0.041	0.064	0.063	0.114	**0.346**	0.214	0.228	0.924	0.842	0.903	0.818	0.517	0.654	**0.597**



(a) Bilibili, betweenness (b) Bilibili, closeness (c) Bilibili, diameter

(d) Weibo, betweenness (e) Weibo, closeness (f) Weibo, diameter

Figure 8: The representative ability of different vertices.

both direct and indirect interactions. The third is local modularity that measures the internal cohesion of community, normalized by the network size. Finally, we adopt conductance to measure the connection sparsity between the community with outside. Smaller conductance means the captured interaction semantics is more complete with clearer community boundary. The above metrics can measure if the community can maintain the network semantics, which is the basis to reflect the network properties.

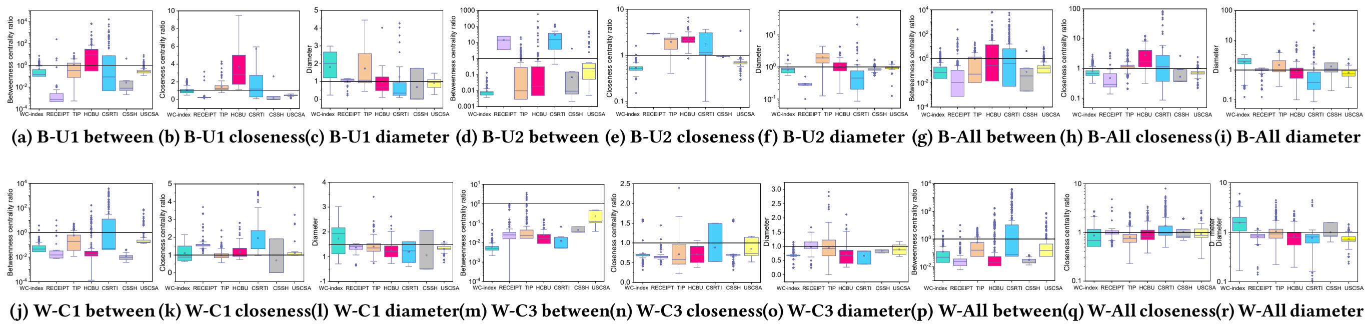
First, the experimental results of the support ratio are shown in Figure 7. We show the average internal support and average external support by different colors in the histogram whose vertical axis is in the left, and show the support ratio by the line chart whose vertical axis is in the right. It is seen that *USCSA* always has larger internal supports and smaller external supports, and its support ratio is the highest on the four datasets. For instance, *USCSA* has average internal supports of 429.84, and average external supports of 83.32 on DBLP dataset. The support ratio of *USCSA* is 67.74, which is 83.63, 40.21, 20.16, 8.26, 1.37 and 1.27 times of that of *WC-index*, *TIP*, *RECEIPT*, *CSRTI*, *CSSH* and *HCBU*, respectively. It indicates that *USCSA* fully captures the direct and indirect interaction semantics. *CSSH* performs poorly on the crawled Bilibili and Weibo datasets. It is because the two networks are star-schema networks, and the meta-paths further extend the sizes of the star clusters in most cases, rather than forming compact closed circulars just like *USCSA* does. *WC-index* and *CSRTI* only focus on the direct interaction, while *RECEIPT*, *HCBU* and *TIP* consider indirect interaction. It is seen that *RECEIPT* with stricter edge-level constraints outperforms *TIP* and *HCBU* with vertex-level constraints on different datasets.

Second, with respect to the three other measurements, the results are shown in Table 2. Please note that the metrics of *RECEIPT*, *HCBU* and *TIP* on Youtube dataset are none since they are only suitable for bipartite graphs. *USCSA* has the best performance on the DBLP dataset and Bilibili dataset. For the Youtube dataset, it is seen from Table 2 that *USCSA* achieves the second highest degree ratio (lower than *CSRTI*), the medium modularity (lower than *CSRTI* and *CSSH*) and the second best conductance (*CSSH* is the best). Since the degree ratio of *USCSA* is close to the highest one of *CSRTI*, the inside relationships within the communities of *USCSA* are much denser than outside. However, the modularity of *USCSA* becomes lower than not only *CSRTI* but also *CSSH*. It indicates that the sizes of the communities obtained by *USCSA* are smaller than *CSRTI* and *CSSH*, thus the modularity of *USCSA* is underestimated when normalized by the network size. On the contrary, the community sizes of *CSSH* are so large that the obtained communities are almost the same for different queries without any uniqueness, leading to high modularity and small conductance. Since modularity is the optimization goal of *CSRTI* in the detection phase, *CSRTI* achieves higher modularity than *USCSA*. But its bigger conductance indicates that the community boundaries of *CSRTI* are not clear, thus the obtained communities can not maintain complete complex network semantics. To sum up, *USCSA* can capture both direct and indirect interaction semantics and have a more complete expression of the network semantics on the Youtube dataset.

For the Weibo dataset, *USCSA* has a medium degree ratio, second largest modularity and the best conductance. By analyzing the communities of *USCSA*, the degree ratios of the content vertices are the highest, but those of the user vertices are low. The reason is as below. Besides the interested topics, users also focus on various popular topics in the hotspot list everyday on Weibo, thus the degree ratios of user vertices are low. Even though, *USCSA* performs well in modularity and conductance, i.e., It can find the interest communities with cohesion guarantee and clear boundaries.

In summary, *USCSA* achieves effectiveness in community searching on different datasets. *USCSA* can deeply mine the complex network semantics by using the (τ, ρ)-camp community model, and effectively search the cohesive communities.

6.4.2 Representative ability evaluation. This experiment evaluates the ability of the communities to represent the centrality property and connectivity property of the network. We adopt the betweenness centrality, closeness centrality and diameter [18] as the metrics to evaluate the properties. Betweenness centrality reflects the bridge role of a vertex based on the shortest paths. Closeness centrality is a measure of how central a vertex is within a network. Diameter is the longest shortest distance between a vertex with others. We


(a) B-U1 between (b) B-U1 closeness (c) B-U1 diameter (d) B-U2 between (e) B-U2 closeness (f) B-U2 diameter (g) B-All between (h) B-All closeness (i) B-All diameter

(j) W-C1 between (k) W-C1 closeness (l) W-C1 diameter (m) W-C3 between (n) W-C3 closeness (o) W-C3 diameter (p) W-All between (q) W-All closeness (r) W-All diameter

Figure 9: The representative ability of different methods ('B': Bilibili, 'W': Weibo, and "between" is short for betweenness).

compute the metric value of each vertex in the community, as well as the metric value of this vertex in the network, and compute the ratio between them. To distinguish the roles of different vertices, the user vertices are decomposed into 4 buckets by partitioning the degree range in average, and it is similar for the content vertices.

At first, we analyze the roles of the vertices in different buckets. Taking Bilibili and Weibo datasets for example, Figure 8 shows the distribution of the three metrics on the 8 buckets by the box plot, where U1 is the first user bucket and C1 is the first content bucket, etc. In the box plot, the box represents the interquartile range (IQR), i.e., the range between lower quartile and upper quartile, and the middle line represents the medium. From U1 to U4 or from C1 to C4, it is seen that the mediums of the betweenness centrality and closeness centrality increase, and the distributions are more concentrated. It indicates that as the degree becomes larger, the vertex is more central and connects more pairs of vertices. In terms of diameter, the medium becomes smaller and the distribution is also more concentrated, consistent with that of closeness centrality. In summary, the vertices with bigger degrees are more probabilistic to play more important roles in community formation.

Afterwards, to evaluate the ability of the communities to represent the whole network, we take the U1, U2 buckets on Bilibili dataset and C1, C3 buckets on Weibo dataset for illustration, as well as analyze the whole Bilibili dataset and Weibo dataset. The three metric ratios are compared for different competitors, and the box plots are shown in Figure 9. In each subfigure, the horizontal line of $y = 1$ is drawn. The closer the ratio is to 1, the better representative ability the method has. Moreover, smaller IQR size means that the metric distribution of vertices in the community is more similar with the metric distribution of those vertices in the network.

First, in terms of WC-index, the IQRs on betweenness centrality ratio and closeness centrality ratio are under the line of $y = 1$ for the whole Bilibili dataset and Weibo dataset (Figure 9(g)~(i) and (p)~(r)), while the IQRs on diameter are beyond $y = 1$. It means that *WC-index* stretches the distance between vertices since it aims to find the star-schema communities based on k-core. Second, with respect to *RECEIPT*, *TIP* and *HCBU*, the relative positions of IQRs and the line of $y = 1$ always vary on the three metric ratios in different buckets, and the IQR sizes also fluctuate. It indicates that the methods particularly designed for bipartite graphs can not maintain stable representative performance for different types of vertices. Third, the IQRs of *CSRTI* are always large on different metric ratios, i.e., its representative performance for different queries are quite different. Namely, the performance of *CSRTI* heavily relies on the queries. Forth, the IQR sizes of *CSSH* also fluctuate in different buckets. Since the meta-paths connect almost each pair of vertices, the relationship explosion confuses the network semantics. Finally, the IQRs of *USCSA* are relatively close to the line of $y = 1$ on the three metric ratios. Besides, the IQR sizes of *USCSA* are the smallest among all the methods in most cases, which means the metric distribution of vertices in the community are close to the metric distribution of those vertices in the network. Therefore, *USCSA* achieves good overall representative performance.

In summary, through fully capturing the complex network semantics by maximizing the unified network semantics based on the (τ, ρ)-camp model, *USCSA* can effectively find the communities that can reflect important properties of the network.

7 CONCLUSION

In this paper, we have studied the community search problem with the goal of finding the communities with the densest network semantics to reflect important network properties. We first design a novel community model named (τ, ρ)-camp to capture both direct interaction semantics and indirect interaction semantics in any level of granularity. By exploring the nested property of the communities, a hierarchical index is constructed on the communities for efficient community search. Based on the greedy strategy, we solve the proposed NP-hard community search problem by designing an approximate community search algorithm named *USCSA*, with approximation ratio of 2 and linear time complexity. The experimental results prove that *USCSA* achieves search efficiency and the obtained communities truly reflect the network properties.

ACKNOWLEDGMENTS

This work was supported by NSFC-Regional Joint Fund Key Program (grant no.U22A2036), Research Team Project supported by Natural Science Foundation of Heilongjiang (grant no.TD2022F001), NSFC-Enterprise Joint Fund Key Program (grant no.U20B2048, U21B2019), NSFC-General Program (grant no.62272127), and Natural Science Foundation Joint Guidance Project of Heilongjiang (grant no.LH2024F036).

REFERENCES

[1] Esra Akbas and Peixiang Zhao. 2017. Truss-based community search: a truss-equivalence based indexing approach. *Proceedings of the VLDB Endowment* 10, 11 (2017), 1298–1309.

[2] Zi Chen, Yiwei Zhao, Long Yuan, Xuemin Lin, and Kai Wang. 2023. Index-based biclique percolation communities search on bipartite graphs. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 2699–2712.

[3] James Cheng, Yiping Ke, Shumo Chu, and M Tamer Özsu. 2011. Efficient core decomposition in massive networks. In *2011 IEEE 27th International Conference on Data Engineering*. 51–62.

[4] Jonathan Cohen. 2008. Trusses: Cohesive subgraphs for social network analysis. *National security agency technical report* 16, 3.1 (2008).

[5] Wanyun Cui, Yanghua Xiao, Haixun Wang, and Wei Wang. 2014. Local search of communities in large graphs. In *International Conference on Management of Data, SIGMOD*. ACM, 991–1002.

[6] Yizhou Dai, Miao Qiao, and Rong-Hua Li. 2024. On Density-based Local Community Search. *Proceedings of the ACM on Management of Data* 2, 2 (2024), 1–25.

[7] Mark De Berg. 2000. *Computational geometry: algorithms and applications*. Springer Science & Business Media.

[8] Zheng Dong, Xin Huang, Guorui Yuan, Hengshu Zhu, and Hui Xiong. 2021. Butterfly-Core Community Search over Labeled Graphs. *Proc. VLDB Endow.* 14, 11 (2021), 2006–2018.

[9] Jordi Duch and Alex Arenas. 2005. Community detection in complex networks using extremal optimization. *Physical Review E—Statistical, Nonlinear, and Soft Matter Physics* 72, 2 (2005), 027104.

[10] Yixiang Fang, Xin Huang, Lu Qin, Ying Zhang, Wenjie Zhang, Reynold Cheng, and Xuemin Lin. 2020. A survey of community search over big graphs. *The VLDB Journal* 29 (2020), 353–392.

[11] Yixiang Fang, Yixing Yang, Wenjie Zhang, Xuemin Lin, and Xin Cao. 2020. Effective and Efficient Community Search over Large Heterogeneous Information Networks. *Proc. VLDB Endow.* 13, 6 (2020), 854–867.

[12] Santo Fortunato and Darko Hric. 2016. Community detection in networks: A user guide. *Physics reports* 659 (2016), 1–44.

[13] Johannes Gasteiger, Aleksandar Bojchevski, and Stephan Günnemann. 2018. Predict then propagate: Graph neural networks meet personalized pagerank. *arXiv preprint arXiv:1810.05997* (2018).

[14] Juris Hartmanis. 1982. Computers and intractability: a guide to the theory of np-completeness. *Siam Review* 24, 1 (1982), 90.

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

[16] Xin Huang, Laks VS Lakshmanan, and Jianliang Xu. 2017. Community search over big graphs: Models, algorithms, and opportunities. In *2017 IEEE 33rd international conference on data engineering (ICDE)*. 1451–1454.

[17] Xin Huang, Laks V. S. Lakshmanan, Jeffrey Xu Yu, and Hong Cheng. 2015. Approximate Closest Community Search in Networks. *Proc. VLDB Endow.* 9, 4 (2015), 276–287.

[18] Yangqin Jiang, Yixiang Fang, Chenhao Ma, Xin Cao, and Chunshan Li. 2022. Effective Community Search over Large Star-Schema Heterogeneous Information Networks. *Proc. VLDB Endow.* 15, 11 (2022), 2307–2320.

[19] Yangqin Jiang, Yixiang Fang, Chenhao Ma, Xin Cao, and Chunshan Li. 2022. Effective community search over large star-schema heterogeneous information networks. *Proceedings of the VLDB Endowment* 15, 11 (2022), 2307–2320.

[20] Junghoon Kim, Siqiang Luo, Gao Cong, and Wenyuan Yu. 2022. DMCS: Density modularity based community search. In *Proceedings of the 2022 International Conference on Management of Data*. 889–903.

[21] Kartik Lakhotia, Rajgopal Kannan, Viktor K. Prasanna, and César A. F. De Rose. 2021. RECEIPT: REfine CoarsE-grained IndePendent Tasks for Parallel Tip decomposition of Bipartite Graphs. *Proc. VLDB Endow.* 14, 3 (2021), 404–417.

[22] Xuankun Liao, Qing Liu, Xin Huang, and Jianliang Xu. 2024. Truss-Based Community Search over Streaming Directed Graphs. *Proceedings of the VLDB Endowment* 17, 8 (2024), 1816–1829.

[23] Boge Liu, Fan Zhang, Wenjie Zhang, Xuemin Lin, and Ying Zhang. 2021. Efficient Community Search with Size Constraint. In *37th IEEE International Conference on Data Engineering, ICDE 2021,Chania, Greece, April 19-22, 2021*. IEEE, 97–108.

[24] Fragkiskos D Malliaros, Christos Giatsidis, Apostolos N Papadopoulos, and Michalis Vazirgiannis. 2020. The core decomposition of networks: Theory, algorithms and applications. *The VLDB Journal* 29, 1 (2020), 61–92.

[25] Ahmet Erdem Sarıyüce and Ali Pinar. 2018. Peeling bipartite networks for dense subgraph discovery. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. 504–512.

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

[27] Longxu Sun, Xin Huang, Rong-Hua Li, Byron Choi, and Jianliang Xu. 2022. Index-Based Intimate-Core Community Search in Large Weighted Graphs. *IEEE Trans. Knowl. Data Eng.* 34, 9 (2022), 4313–4327.

[28] Yifu Tang, Jianxin Li, Nur Al Hasan Haldar, Ziyu Guan, Jiajie Xu, and Chengfei Liu. 2022. Reliable community search in dynamic networks. *Proceedings of the VLDB Endowment* 15, 11 (2022), 2826–2838.

[29] Yifu Tang, Jianxin Li, Nur Al Hasan Haldar, Ziyu Guan, Jiajie Xu, and Chengfei Liu. 2024. Reliability-Driven Local Community Search in Dynamic Networks. *IEEE Transactions on Knowledge and Data Engineering* 36, 2 (2024), 809–822.

[30] Jia Wang and James Cheng. 2012. Truss Decomposition in Massive Networks. *Proc. VLDB Endow.* 5, 9 (2012), 812–823.

[31] Kai Wang, Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang. 2022. Towards efficient solutions of bitruss decomposition for large-scale bipartite graphs. *VLDB J.* 31, 2 (2022), 203–226.

[32] Kai Wang, Wenjie Zhang, Xuemin Lin, Ying Zhang, and Shunyang Li. 2022. Discovering Hierarchy of Bipartite Graphs with Cohesive Subgraphs. In *38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022*. IEEE, 2291–2305.

[33] Kai Wang, Wenjie Zhang, Xuemin Lin, Ying Zhang, Lu Qin, and Yuting Zhang. 2021. Efficient and effective community search on large-scale bipartite graphs. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 85–96.

[34] Xiaoqin Xie, Shuangyuan Liu, Jiaqi Zhang, Shuai Han, Wei Wang, and Wu Yang. 2024. Efficient Community Search Based on Relaxed k-Truss Index. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 1691–1700.

[35] Yichen Xu, Chenhao Ma, Yixiang Fang, and Zhifeng Bao. 2024. Efficient and effective algorithms for densest subgraph discovery and maintenance. *VLDB J.* 33, 5 (2024), 1427–1452.

[36] Jaewon Yang and Jure Leskovec. 2012. Defining and evaluating network communities based on ground-truth. In *Proceedings of the ACM SIGKDD workshop on mining data semantics*. 1–8.

[37] Junhao Ye, Yuanyuan Zhu, and Lu Chen. 2023. Top-r keyword-based community search in attributed graphs. In *39th IEEE International Conference on Data Engineering, ICDE 2023,Anaheim, CA, USA, April 3-7, 2023*. IEEE, 1652–1664.

[38] Long Yuan, Lu Qin, Wenjie Zhang, Lijun Chang, and Jianye Yang. 2019. Index-Based Densest Clique Percolation Community Search in Networks. In *35th IEEE International Conference on Data Engineering*. IEEE, 2161–2162.

[39] Chen Zhang, Fan Zhang, Wenjie Zhang, Boge Liu, Ying Zhang, Lu Qin, and Xuemin Lin. 2020. Exploring Finer Granularity within the Cores: Efficient (k, p)-Core Computation. In *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020*. IEEE, 181–192.

[40] Fan Zhang, Haicheng Guo, Dian Ouyang, Shiyu Yang, Xuemin Lin, and Zhihong Tian. 2024. Size-Constrained Community Search on Large Networks: An Effective and Efficient Solution. *IEEE Trans. Knowl. Data Eng.* 36, 1 (2024), 356–371.

[41] Fan Zhang, Wenjie Zhang, Ying Zhang, Lu Qin, and Xuemin Lin. 2017. OLAK: An Efficient Algorithm to Prevent Unraveling in Social Networks. *Proc. VLDB Endow.* 10, 6 (2017), 649–660.

[42] Yun Zhang, Charles A Phillips, Gary L Rogers, Erich J Baker, Elissa J Chesler, and Michael A Langston. 2014. On finding bicliques in bipartite graphs: a novel algorithm and its application to the integration of diverse biological data types. *BMC bioinformatics* 15 (2014), 1–18.

[43] Yingli Zhou, Yixiang Fang, Wensheng Luo, and Yunming Ye. 2023. Influential Community Search over Large Heterogeneous Information Networks. *Proc. VLDB Endow.* 16, 8 (2023), 2047–2060.