




The art of characterization in large networks: Finding the critical attributes

Renjie Sun¹ · Chen Chen¹  · Xiaoyang Wang¹ · Yanping Wu¹ · Mengqi Zhang¹ · Xijuan Liu¹

Received: 18 March 2021 / Revised: 19 May 2021 / Accepted: 1 June 2021 /
Published online: 11 June 2021

© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2021

Abstract

Recently, with the development of online social networks, users in social networks are usually associated with attributes such as user preferences, which is of great importance for analyzing the properties of social networks. To identify critical attributes, we propose and investigate a new problem named attribute k -core maximization. Given an attribute graph G and a budget b , we aim to identify a set of b attributes, such that the corresponding attribute k -core is maximized. Due to the NP-hardness of the problem, we resort to the greedy strategy in this paper. In order to handle large graphs, a layer-based structure and novel searching paradigms are developed to accelerate the computation. Finally, experiments over 6 real-world networks are conducted to evaluate the performance of proposed model and techniques.

Keywords Attribute graph · Attribute k -core · Maximization · NP-hard

This article belongs to the Topical Collection: *Special Issue on Large Scale Graph Data Analytics*
Guest Editors: Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang

✉ Chen Chen
chenc@zjgsu.edu.cn

Renjie Sun
renjiesun.zjgsu@gmail.com

Xiaoyang Wang
xiaoyangw@zjgsu.edu.cn

Yanping Wu
yanpingw.zjgsu@gmail.com

Mengqi Zhang
mengqiz.zjgsu@gmail.com

Xijuan Liu
liuxijuan@zjgsu.edu.cn

¹ Zhejiang Gongshang University, Hangzhou, China

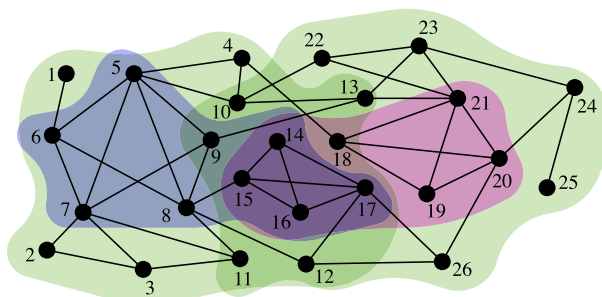
1 Introduction

In real-life applications, graphs are widely adopted to model the relationships among different entities [15, 16]. Mining cohesive subgraphs is one of the most fundamental graph problems in graph analysis [13, 18]. In the literature, different cohesive subgraph models are developed, such as k -core [1], k -truss [22], clique [14], etc. In this paper, we utilize the k -core model to measure the engagement of users in a network. Generally, the k -core of a graph is the maximal subgraph where each vertex inside has at least k neighbors. In real scenarios such as social networks, users in the networks are associated with attributes (e.g., personal preferences) [4, 9]. In attribute graphs, a user is more likely to build contacts with neighbors who share common interests with him. For example, in social networks, friends with common hobby of sport will have greater potential to join a sport community together.

Motivated by this, in this paper, we develop a new model named attribute k -core to better characterize a community. Given an attribute graph and a subset of attribute λ , a maximal subgraph is the attribute k -core depending on λ if each vertex has at least k neighbors and each neighbor shares at least one common attribute in λ with the vertex. As shown in the case study (Figure 8) in the experiment, different selected attributes will lead to different communities. To compute the attribute k -core, we can iteratively remove the vertices that violate the constraints. To find the critical attributes and characterize the main properties of the network, we propose and investigate the attribute k -core maximization problem. Specifically, given an attribute graph and a budget b , we aim to identify a set of b attributes that can lead to the largest attribute k -core.

Example 1 Figure 1 shows a small network with 26 users and their corresponding attributes. Suppose $k = 3$ and $b = 1$. $\{u_5, u_6, \dots, u_9, u_{14}, \dots, u_{17}\}$ is the corresponding attribute 3-core induced by attribute a_1 . $\{u_{14}, \dots, u_{17}, u_{18}, \dots, u_{21}\}$ is the corresponding attribute 3-core induced by attribute a_2 . $\{u_{18}, u_{19}, u_{20}, u_{21}\}$ is the corresponding attribute 3-core induced by attribute a_3 . Hence, attribute a_1 is the optimal result for budget 1.

In the literature, a lot of research are conducted over attribute graphs for cohesive subgraph analysis, e.g., [4, 7, 9, 11]. However, most of them focus on identifying the cor-



a_1	$u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8, u_9, u_{10}, u_{11}, u_{12}, u_{13}, u_{14}, u_{15}, u_{16}, u_{17}, u_{18}$
a_2	$u_9, u_{10}, u_{11}, u_{12}, u_{13}, u_{14}, u_{15}, u_{16}, u_{17}, u_{18}, u_{19}, u_{20}, u_{21}, u_{22}, u_{23}, u_{24}, u_{25}, u_{26}$
a_3	$u_4, u_5, u_8, u_9, u_{12}, u_{17}, u_{18}, u_{19}, u_{20}, u_{21}, u_{23}, u_{25}, u_{26}$

Figure 1 Motivating example (The table presents the attributes that each vertex associates with)

responding community for a given attribute query (e.g., [9, 11]), or finding the attribute association in the graph (e.g., [7]). In addition, they usually emphasize that each vertex should contain the query attributes instead of enforcing the share of common attributes between neighbors. The attribute k -core maximization problem can find many applications. For video game design, the company can use the identified attributes as product features, especially for the games that involve multi-gamers cooperations. Similarly, for party hosting or conference organization, the found attributes can serve as the party or conference theme, which have greater potential to attract more attendances and build a more harmonious communication atmosphere.

Challenges and Contributions To the best of our knowledge, this is the first work to investigate the attribute k -core maximization problem. Although we can compute the attribute k -core in linear time, the attribute k -core maximization problem is NP-hard. Due the large search space, in this paper, we employ the greedy strategy by iteratively selecting the best attribute. However, the number of attributes in real-life networks is usually large. In the greedy framework, a major cost is to compute the marginal gain of adding a new attribute. To scale for large networks, we propose a layer-based method and the corresponding pruning strategies to reduce the exploration space. Furthermore, novel searching paradigms are developed to compute the marginal gain. Finally, we conduct experiments on 6 real networks to verify the advantages of our developed techniques.

Roadmap The rest of the paper is organized as follows. In Section 2, we formally define the problem and prove its properties. The baseline method and optimized solution are presented in Sections 3 and 4, respectively. We report the experiment results in Section 5. Lastly, we review the related work in Section 6 and conclude the paper in Section 7.

2 Preliminaries

In this section, we first introduce some related concepts and formally define problem investigated. Then, we prove the hardness of the problem and show its properties. Mathematical notations that are frequently used throughout this paper are summarized in Table 1.

2.1 Problem definition

We consider an undirected network $G = (V, E, \Lambda)$ as an attribute graph, where V (resp. E) represents the set of vertices (resp. edges) in G and $\Lambda = \{a_1, a_2, \dots, a_t\}$ represents the set of all attributes in G . Each vertex $u \in V$ is associated with a set of attributes, denoted by $\Lambda(u) \subseteq \Lambda$. $n = |V|$ and $m = |E|$ represent the number of vertices and edges in G , respectively. Given an attribute subset $\lambda \subseteq \Lambda$, we use $V(\lambda)$ to denote the set of vertices that contain at least one attribute in λ , i.e., $\forall v \in V(\lambda), \Lambda(v) \cap \lambda \neq \emptyset$. Given an attribute graph G , a subgraph $S = (V_S, E_S, \Lambda_S)$ is an induced subgraph of G , if $V_S \subseteq V$, $E_S = E \cap (V_S \times V_S)$ and $\Lambda_S \subseteq \Lambda$. Given a vertex u , we use $N(u, S)$ to denote the set of its neighbors in S , and $d(u, S) = |N(u, S)|$ to denote its degree in S .

Definition 1 (k -core) Given a graph G and a positive integer k , an induced subgraph $S \subseteq G$ is the k -core of G , denoted by $C_k(G)$, if it satisfies the following constraints. *i*) $d(u, S) \geq k$ for each vertex $u \in V_S$; *ii*) S is maximal, i.e., any supergraph $S' \supset S$ is not a k -core.

Table 1 Summary of notations

Notation	Definition
G	an unweighted undirected attribute graph
V, E	vertex set and edge set in G
Λ	all attributes in G
λ, λ_0	the subset of Λ
u, v, w	vertex in G
a, a_i	attribute in G
$\Lambda(u)$	attribute set of vertex u
$V(\{a\}), V(\lambda_0)$	the set of vertices that contain a / at least one attribute in λ_0
S	the subgraph of G
$N(uS)$	the set of u 's neighbors in S
$d(uS)$	the degree of u in S
$C_k(G)$	the k -core of G
$\widehat{N}(u, S, \lambda), \widehat{N}(u, \lambda)$	the resonant neighbor set of u in S depending on λ
$\widehat{d}(u, S, \lambda), \widehat{d}(u, \lambda)$	the resonant degree of u in S
$\widehat{C}_k(G, \lambda), \widehat{C}_k(\lambda)$	the attribute k -core of G depending on λ
L_λ, L_a	the layer structure of vertices in $V(\lambda) / V(\{a\})$
$l_\lambda(u), l_a(u)$	the layer number of u in L_λ / L_a
$d^+(u)$	the degree upper bound of u in $\widehat{C}_k(G, \lambda)$
$nd^+(u)$	the new degree upper bound of u in $\widehat{C}_k(G, \lambda)$

To compute the k -core, we can iteratively delete the node that violates the degree constraint, which time complexity is $\mathcal{O}(m)$ [1]. Based on the k -core model, we can measure the cohesiveness of a community. However, in real applications, such as social networks,

users with common hobbies are usually closer to each other than others, because it is easier for them to build contacts through the shared interests.

To better model the communities in attribute graphs, we introduce the concept of resonant degree.

Then, we present the formal definition of attribute k -core based on resonant degree.

Definition 2 (resonant degree) Given a graph G and a subset of attributes $\lambda \in \Lambda$, let S be the subgraph induced by $V(\lambda)$. For $u \in V_S$, we use $\widehat{N}(u, S, \lambda)$ to denote the resonant neighbor set of u in S depending on λ , such that each resonant neighbor of u has at least one common attribute with u , i.e., $\forall v \in \widehat{N}(u, S, \lambda)$ satisfies that $\Lambda(u) \cap \Lambda(v) \neq \emptyset$. The resonant degree of u in S equals $|\widehat{N}(u, S, \lambda)|$, denoted by $\widehat{d}(u, S, \lambda)$.

Definition 3 (attribute k -core) Given an attribute graph G and a subset of attributes $\lambda \in \Lambda$, an induced subgraph S is the attribute k -core of G depending on λ , denoted by $\widehat{C}_k(G, \lambda)$, if it satisfies *i*) each vertex $u \in S$ has at least k resonant neighbors, i.e., $\widehat{d}(u, S, \lambda) \geq k$; and *ii*) S is maximal, i.e., any superset of it is not an attribute k -core.

For the simplicity, when the context is clear, we use $\widehat{N}(u, \lambda)u\lambda$, $\widehat{d}(u, \lambda)u\lambda$ and $\widehat{C}_k(\lambda)\lambda$ instead of $\widehat{N}(u, S, \lambda)$, $\widehat{d}(u, S, \lambda)$ and $\widehat{C}_k(G, \lambda)$, respectively. In addition, we use degree

and resonant degree interchangeably. We use $|\widehat{C}_k(\lambda)\lambda|$ to denote the number of vertices in $\widehat{C}_k(\lambda)\lambda$.

Obviously, different selected attribute sets can lead to different attribute k -cores. In this paper, in order to analyze the properties of the given attribute graph, we aim to find an attribute set λ^* to maximize the size of the corresponding attribute k -core.

Problem Statement Given an attribute graph $G = (V, E, A)$, two positive integers k and b , the **attribute k -core maximization** problem aims to identify a set λ^* of b attributes from A , such that the size of corresponding attribute k -core is maximized, i.e.,

$$\lambda^* = \arg \max_{\lambda \subseteq A \wedge |\lambda|=b} |\widehat{C}_k(\lambda)\lambda|$$

2.2 Problem properties

According to Theorems 1 and 2, the investigated problem is NP-hard, and the object function is monotone but not submodular.

Theorem 1 *Given an attribute graph G , the attribute k -core maximization problem is NP-hard for any k .*

Proof When $k \geq 1$, we reduce the maximum coverage problem [6] to the attribute k -core maximization problem. The maximum coverage problem that is given several sets and a number b , each set may have several same elements, we must select at most b sets such that the maximum number of elements are covered, i.e., the union of the selected sets has the largest size. We consider an arbitrary instance of the maximum coverage problem with s sets T_1, T_2, \dots, T_s and t elements $\{e_1, e_2, \dots, e_t\} = \bigcup_{1 \leq i \leq s} T_i$. Then we construct a corresponding instance of the attribute k -core maximization problem in an attribute graph G as follows.

In this attribute graph G , the attribute set of this graph is A with s attributes $\{a_1, a_2, \dots, a_s\}$, where each attribute a_i corresponds to the set T_i for any $1 \leq i \leq s$. The set of vertices in G consists of two parts: M and P . M consist of $k + 1$ vertices in which every pair of vertices in M are adjacent, and the associated attribute set of every vertex in M is A . P consist of t parts P_1, P_2, \dots, P_t , where each part P_j ($1 \leq j \leq t$) corresponds to the elements e_j and P_j consists of k vertices $p_{j,1}, p_{j,2}, \dots, p_{j,k}$. For each P_j ($1 \leq j \leq t$) we add an edge for each pair of vertices such that each P_j is a k -clique. At this stage, the degree of each vertex in P is $k - 1$. Next, we add an edge from each vertex in P to any vertex in M to make sure the degree of each vertex in P is exactly k . Then, we add attributes for each vertex in P . If T_i consist of e_j , we add attribute a_i to the associated attribute set of each vertex in P_j . Figure 2 is an example of the attribute G with $k = 3$ constructed from 4 sets and 4 elements.

With the construction, we ensure that (i) none of vertex in M will be deleted, because the number of resonant neighbors of each vertex in M will not less than k no matter what attribute set we choose from A ; (ii) the degree of all vertices in P is exactly equals to k ; (iii) P_j will be deleted unless we select the corresponding attribute; and (iv) all P_j have the same size for $1 \leq j \leq t$. By doing this, the optimal solution of the attribute k -core maximization problem corresponds to the optimal solution of the maximum coverage problem. Since the maximum coverage problem is NP-hard, we prove that the attribute k -core maximization problem is NP-hard for any $k \geq 1$. \square

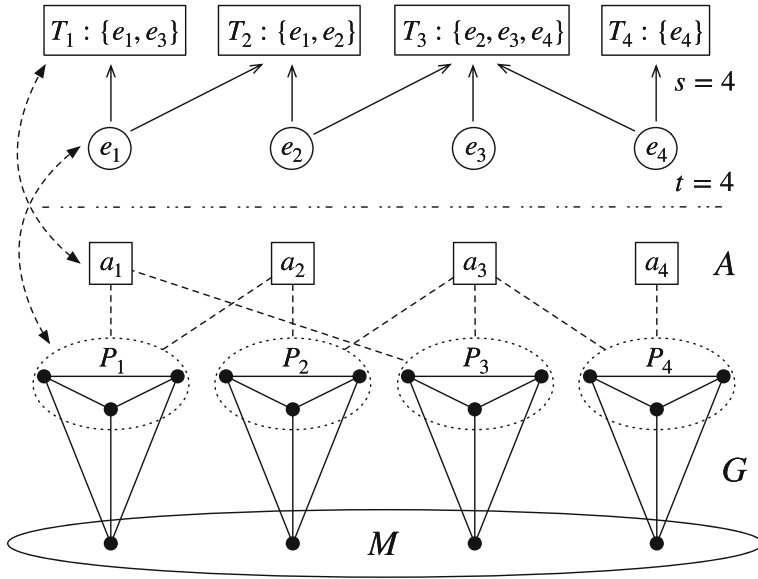


Figure 2 Example of NP-hard proof

Theorem 2 The object function $f(\lambda) = |\widehat{C}_k(\lambda)\lambda|$ is monotonic but not submodular.

Proof Monotonic. Suppose there is an attribute set $\lambda_0 \subset \lambda'_0$. The attribute k -core $\widehat{C}_k(\lambda)\lambda_0$ must be contained in $\widehat{C}_k(\lambda)\lambda'_0$, because adding any attribute in $\lambda'_0 \setminus \lambda_0$ can not decrease the degree of $u \in \widehat{C}_k(\lambda)\lambda_0$. Thus, $f(\lambda_0) \leq f(\lambda'_0)$ and f is monotonic.

Non-submodular The function f is submodular if $f(\lambda_1 \cup \lambda_2) + f(\lambda_1 \cap \lambda_2) \leq f(\lambda_1) + f(\lambda_2)$ for any attribute subsets. We prove the theorem by construct a counter example. As shown in Figure 1, for $k = 3$, suppose $\lambda_1 = \{a_1\}$ and $\lambda_2 = \{a_2\}$. We have $f(\lambda_1) = 9$, $f(\lambda_2) = 8$, $f(\lambda_1 \cup \lambda_2) = 13$ and $f(\lambda_1 \cap \lambda_2) = 0$. The inequation does not hold. Thus, function f is non-submodular. □

Algorithm 1 Baseline algorithm.

Input : G : an attribute graph, k : resonant degree constraint, b : number of selected attributes

Output: λ : the set of b attributes

- 1 $G \leftarrow C_k(G)$;
 - 2 $\lambda \leftarrow \emptyset$;
 - 3 **while** $|\lambda| < b$ **do**
 - 4 $a^* \leftarrow \arg \max_{a \in \Lambda \setminus \lambda} |\widehat{C}_k(\lambda \cup \{a\})|$;
 - 5 $\lambda \leftarrow \lambda \cup \{a^*\}$;
 - 6 **return** λ
-

3 Baseline algorithm

A naive solution for the problem is to enumerate all the possible attribute sets with size b and return the optimal result. The time complexity is $\mathcal{O}\left(\binom{|A|}{b}m\right)$, which is not affordable for real-life networks. Considering the NP-hardness of the problem, we resort to the greedy heuristic by iteratively choosing the best attribute. Besides, as the monotonic property discussed in Theorem 2, the size of attribute k -core $|\widehat{C}_k(\lambda)\lambda|$ is positively correlated with $|\lambda|$, which means $\widehat{C}_k(\lambda)\lambda$ will be expanded with new attribute added into λ . Details are shown in the following theorem.

Theorem 3 *Suppose that $\lambda \subseteq \lambda'$ is the currently selected attribute set. If vertex u belongs to $\widehat{C}_k(\lambda)\lambda$, u must be in $\widehat{C}_k(\lambda)\lambda'$.*

Proof We have that $\lambda \subseteq \lambda'$. Therefore, $\widehat{C}_k(\lambda)\lambda$ must be contained in $\widehat{C}_k(\lambda)\lambda'$ according to Theorem 2. \square

The baseline greedy method is shown in Algorithm 1. We first compute the k -core of G in Line 1 since any attribute k -core must be inside the k -core of G .

Then, we initialize an empty set λ to store all the selected attributes (Line 2). At each iteration, we compute the corresponding attribute k -core and choose the best attribute whose addition can enlarge the size of attribute k -core most (Lines 4-5). The algorithm terminates when b attributes are selected. The time complexity is $\mathcal{O}(bm|A|)$, which significantly reduces the computation cost compared with the native approach.

4 Optimized solution

As shown in the experiment results, the baseline greedy method can greatly accelerate the search with competitive results. However, it is still cannot scale for large networks. In this section, we first present a novel core-spread searching framework and then introduce some pruning techniques to reduce the candidate space.

4.1 Core-spread framework

As observed in the greedy framework, a major cost is to compute the marginal gain of adding an attribute to the currently selected attribute set. Therefore, we introduce the core-spread framework to facilitate the computation. Before presenting the details of the framework, we first apply the example in Figure 3 to clarify some notations and concepts. Suppose the currently selected attribute set is λ_0 and the new added attribute is a .

The left (resp. right) dotted ellipse is $V(\lambda_0)$ (resp. $V(\{a\})$), and the left (resp. right) solid ellipse represents $\widehat{C}_k(\lambda)\lambda_0$ (resp. $\widehat{C}_k(\lambda)\{a\}$). Note that, we use $\mathcal{UV} = V(\lambda_0) \cup V(\{a\})$ (resp. $\mathcal{IV} = V(\lambda_0) \cap V(\{a\})$) to represent the union (resp. intersection) of $V(\lambda_0)$ and $V(\{a\})$. Similarly, we use $\mathcal{UC} = \widehat{C}_k(\lambda)\lambda_0 \cup \widehat{C}_k(\lambda)\{a\}$ (resp. $\mathcal{IC} = \widehat{C}_k(\lambda)\lambda_0 \cap \widehat{C}_k(\lambda)\{a\}$) to represent the union (resp. intersection) of $\widehat{C}_k(\lambda)\lambda_0$ and $\widehat{C}_k(\lambda)\{a\}$. When adding a new attribute a to λ_0 , it is obvious that vertices in \mathcal{IV} may have their resonate degree changed and even affect other vertices, such that some vertices will join the updated attribute k -core, i.e., $\widehat{C}_k(\lambda)\lambda_0 \cup \{a\}$. Hence, if we can efficiently calculate the update degree for promising vertices instead of computing the attribute k -core from scratch, we can save a lot of computation.

In our core-spread framework, each vertex in \mathcal{UV} has three statuses. The vertex is **explored** if it has been checked. If the vertex satisfies the resonate degree constraint, its status is **survived**, otherwise it is marked as **deleted**. A deleted vertex will not involve in the enlarged attribute k -core, and a currently survived vertex may be deleted later after further verification. In this paper, we derive the upper bound $d^+(u)$ of u 's degree to quickly filter unpromising ones. Specifically, $d^+(u)$ is the number of vertices in the union of u 's survived neighbors, unexplored neighbors and all neighbors in \mathcal{UC} . We will remove u if $d^+(u) < k$. In the framework, we traverse from all vertices of \mathcal{TV} , which are called **source vertices**. With the addition of a new attribute, some source vertices will preserve in the updated attribute k -core, which further facilitates some originally unsatisfied vertices retain in the final attribute k -core.

Algorithm 2 THE CORE-SPREAD FRAMEWORK.

```

Input   :  $G$  : an attribute graph,  $k$  : resonate degree constraint,  $\lambda_0$  : the currently selected
            attribute set,  $a$  : newly added attribute
Output :  $\widehat{C}_k(\lambda_0 \cup \{a\})$  : the updated attribute  $k$ -core after adding  $a$ 
1 foreach  $u \in \mathcal{UV}$  do
2    $e(u) \leftarrow 0, s(u) \leftarrow 0$ ;
3 foreach  $u \in \mathcal{UC}$  do
4    $e(u) \leftarrow 1, s(u) \leftarrow 1, d^+(u) \leftarrow +\infty$ ;
5  $\mathcal{H} \leftarrow \mathcal{TV}$ ;
6 while  $\mathcal{H} \neq \emptyset$  do
7    $u \leftarrow \mathcal{H}.pop()$ ;
8    $e(u) \leftarrow 1$ ;
9   Compute  $d^+(u)$ ;
10  if  $d^+(u) \geq k$  then
11    $s(u) \leftarrow 1$ ;
12   foreach  $v \in \widehat{N}(u, \lambda_0 \cup \{a\})$  do
13     if  $v \notin \mathcal{H}$  and  $e(v) = 0$  then
14        $\mathcal{H}.push(v)$ ;
15  else
16    $s(u) \leftarrow 0$ ;
17   SHRINK( $u$ );
18 return all the survived vertices
  
```

/* Algorithm 3 */;

The core-spread framework Algorithm 2 illustrates the pseudocode of the framework.

At first, for each vertex $u \in \mathcal{UV}$, the explored value $e(u)$ and the survived value $s(u)$ are both initialized as 0 (Lines 1-2). For each vertex $u \in \mathcal{UC}$, we initialize its explored value and survived value as 1, its upper bound degree as $+\infty$ in Lines 3-4. This is because the vertices in \mathcal{UC} must exist in the corresponding attribute k -core. Then, we put all vertices in \mathcal{TV} , i.e., source vertices, into a queue \mathcal{H} (Line 5), and we visit the vertices in \mathcal{H} iteratively. For each processed vertex u , we set $e(u)$ as 1 and compute its upper bound degree in Lines 8-9. If $d^+(u) \geq k$, its survived value is set as 1 (Lines 10-11) and its unexplored neighbors will be pushed into \mathcal{H} (Lines 12-14). Otherwise, u is not survived due to the violation of degree constraint, and we invoke SHRINK algorithm to update the upper bound degree of u 's neighbors in Lines 15-17. Finally, we return all the survived vertices as the result in Line 18.

Shrink procedure Algorithm 3 shows the details of SHRINK process for a checked vertex u , which aims to update the upper bound degree of u 's neighbors considering the removal of u . We firstly initialize a vertex set \mathcal{T} as \emptyset in Line 1. Then we update the upper bound degree of its survived neighbors and check neighbors' new upper bound degree in Lines 2–5. Specifically, the upper bound degree of its neighbors is reduced by 1 (Line 3). We put the vertices that violate the upper bound degree constraint into \mathcal{T} (Lines 4–5). For each vertex in \mathcal{T} , we set its survived value as 0 and process it by recursively invoking the SHRINK process (Lines 6–8).

Algorithm 3 SHRINK(u).

Input : u : vertex from Algorithm 2

- 1 $\mathcal{T} \leftarrow \emptyset$;
- 2 **foreach** $v \in \widehat{N}(u, \lambda_0 \cup \{a\})$ and $s(v) = 1$ **do**
- 3 $d^+(v) \leftarrow d^+(v) - 1$;
- 4 **if** $d^+(v) < k$ **then**
- 5 $\mathcal{T} \leftarrow \mathcal{T} \cup \{v\}$;
- 6 **foreach** $v \in \mathcal{T}$ **do**
- 7 $s(v) \leftarrow 0$;
- 8 SHRINK(v);

Discussion Based on the core-spread framework, we can speedup the computation of marginal gain when adding a new attribute. However, it still has some drawbacks: *i*) the upper bound is not tight, and *ii*) the searching cost is large considering the large number of attributes. Therefore, in the following sections, optimized upper bound and novel search paradigm are developed to further accelerate the processing.

4.2 Layer-based optimization

Algorithm 4 LAYER CONSTRUCTION(G, λ, k).

Input : G : an attribute graph, λ : the currently selected attribute set, k : resonate degree constraint

Output : layer structure L_λ and the attribute k -core $\widehat{C}_k(\lambda)$

- 1 $i \leftarrow 1$; $\mathcal{N} \leftarrow V(\lambda)$;
- 2 **while** exist vertex $u \in \mathcal{N}$ with $\widehat{d}(u, \lambda) < k$ **do**
- 3 $L_\lambda^i \leftarrow \{u \mid \widehat{d}(u, \lambda) < k \text{ and } u \in \mathcal{N}\}$;
- 4 $\mathcal{N} \leftarrow \mathcal{N} \setminus L_\lambda^i$;
- 5 $i \leftarrow i + 1$;
- 6 $L_\lambda^{+\infty} \leftarrow \mathcal{N}$;
- 7 **return** L_λ, \mathcal{N}

In the core-spread framework, we leverage the degree upper bound to filter the unpromising vertices. Apparently, a tighter and efficient-computed upper bound could accelerate the processing. In this subsection, we employ a layer structure to speedup the computation and derivation of bound. Given a set λ of attributes, the layer structure L_λ organizes the vertices in $V(\lambda)$ following the core decomposition procedure. That is, we iteratively peel the vertices

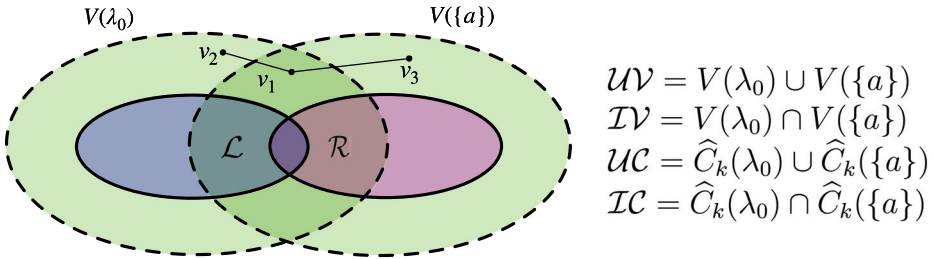


Figure 3 Example of adding attribute a into λ_0

that violate the attribute k -core constraint and store them in the same layer. The rest vertices are processed in the same manner until all the vertices are maintained in the corresponding layers.

Layer construction Different attribute sets lead to vertex sets. We construct the layer structure for each candidate attribute a_i and the currently selected attribute set λ_0 . The details of layer construction are shown in Algorithm 4. We initialize $i = 1$ to denote the number of layer and store all the vertices associated with attribute set λ into \mathcal{N} (Line 1). We process the vertices in \mathcal{N} iteratively. In each iteration, we store all the vertices that violate the degree constraint currently in the same layer L_λ^i (Line 3). Then, we remove L_λ^i from \mathcal{N} . The iteration terminates when all the vertices in \mathcal{N} satisfy the degree constraint. The subgraph induced by the remained vertices in \mathcal{N} is the corresponding attribute k -core $\widehat{C}_k(\lambda)\lambda$, and we store the remained vertices in layer $L_\lambda^{+\infty}$ (Line 6). Eventually, we return L_λ and \mathcal{N} in Line 7.

For a given vertex u , we use $l_\lambda(u)$ to denote its layer number in L_λ . The layer construction phase follows the procedure of k -core decomposition, which time complexity is $\mathcal{O}(m)$. Note that, we need to construct the layer for each attribute, which will be done in the first iteration of the greedy framework.

Following is a layer construction example.

Example 2 Reconsidering the graph in Figure 1, suppose the currently selected attribute set $\lambda_0 = \{a_1\}$ and the verified attribute $a = a_2$. Then, we have $\widehat{C}_3(\lambda_0) = \{v_5, \dots, v_9, v_{14}, \dots, v_{17}\}$ and $\widehat{C}_3(\{a\}) = \{v_{14}, \dots, v_{21}\}$. The constructed layer structures for λ_0 and a are shown in Figure 4. Note that, for the last layer (i.e., $+\infty$ layer), we only draw partial vertices. Initially, for the vertices in $V(\lambda_0)$, the resonant degree of $v_1, v_2, v_{12}, v_{13}, v_{18}$ are less than 3. Thus, we put them in $L_{\lambda_0}^1$ and remove them from $V(\lambda_0)$. Then, the resonant degree of v_3, v_4, v_{10} are smaller than 3, and we have $L_{\lambda_0}^2 = \{v_3, v_4, v_{10}\}$. Moreover, we put v_{11} into the third layer, i.e., $l_{\lambda_0}(v_{11}) = 3$. After deleting v_{11} from $V(\lambda_0)$, the remained vertices correspond to that of $\widehat{C}_3(\lambda_0)$, and we put them into the $+\infty$ layer. Similarly, we build the layer structure L_a for attribute a , which is shown in the right part of Figure 4. For all vertices in $V(\{a\})$, we can use same method to build $L_a^1 = \{v_9, \dots, v_{12}, v_{25}\}$, $L_a^2 = \{v_{13}, v_{22}, v_{24}, v_{26}\}$ and $L_a^3 = \{v_{23}\}$.

Given the currently selected attributes set λ_0 and the newly added attribute a , it is easy to find that many vertices can co-exist in both layer structures, i.e., L_{λ_0} and L_a . According to the layer construction procedure, the vertices in lower layer can provide degree support for the vertices in higher layer, even may make them join the corresponding attribute k -core.

Therefore, during the traversal, we need to consider the impact of neighbors of the currently processed vertex with higher layer value. Furthermore, the vertices in lower layer cannot be ignored directly. Reconsider the example in Figure 4. For two vertices v_{10} and v_{13} , in $V(\lambda_0)$, we have $l_{\lambda_0}(v_{10}) > l_{\lambda_0}(v_{13})$, but $l_a(v_{10}) < l_a(v_{13})$ in L_a . Therefore, they can affect each other. The details can be explained by the concept and theorem of active-path, which are shown as follows.

Definition 4 (active-path) Given an attribute set λ_0 and an attribute a , we say there is an active-path from a source vertex u to another vertex v , if for each two consecutive vertices x and y (they must be neighbor) along this path satisfying $l_a(y) > l_a(x)$ or $l_{\lambda_0}(y) > l_{\lambda_0}(x)$.

Theorem 4 For each vertex $u \in \mathcal{UV} \setminus \mathcal{UC} \setminus \mathcal{IV}$, it may join the updated attribute k -core iff there exists a vertex v in \mathcal{IV} that can form an active-path with u (i.e., $v \rightsquigarrow u$).

Proof Suppose the currently selected attribute set is λ_0 and the added attribute is a . The vertices in $\mathcal{UV} \setminus \mathcal{UC} \setminus \mathcal{IV}$ can be divided into two partitions, $V(\{a\}) \setminus V(\lambda_0) \setminus \mathcal{UC}$ and $V(\lambda_0) \setminus V(\{a\}) \setminus \mathcal{UC}$. We first prove for each vertex $u \in V(\{a\}) \setminus V(\lambda_0) \setminus \mathcal{UC}$. All the neighbors of u can be divided into two sets N_1 and N_2 , where the layer value of vertex in N_1 is lower than $l_a(u)$ and the layer value of vertex in N_2 is no smaller than $l_a(u)$. Clearly, the size of N_2 is less than k . If there is no vertex in \mathcal{IV} can form an active-path with u , all the vertex in N_1 have been deleted when u is processed in the computation of $\widehat{C}_k(\lambda_0 \cup \{a\})$. Therefore, u will be deleted. When $u \in V(\lambda_0) \setminus V(\{a\}) \setminus \mathcal{UC}$, the proof is similar. Thus, this theorem holds. \square

Recall that in the core-spread framework, we conduct the exploration from \mathcal{IV} , and traverse through their neighbor. To further filter the visiting space, we partition the space into three parts. Reconsider the example in Figure 3. For ease of illustration, we use \mathcal{L} (resp. \mathcal{R}) to represent the region of $V(\lambda_0) \cap \mathcal{UC} \setminus \widehat{C}_k(\lambda)\{a\}$ (resp. $V(\{a\}) \cap \mathcal{UC} \setminus \widehat{C}_k(\lambda)\lambda_0$). The partition details and the corresponding theorems are shown as follows.

- Partition 1: The vertices in \mathcal{IC} ;
- Partition 2: The vertices in $\mathcal{R} \cup \mathcal{L}$;
- Partition 3: The vertices in $\mathcal{IV} \setminus (\mathcal{IC} \cup \mathcal{R} \cup \mathcal{L})$.

Theorem 5 In the core-spread framework, we do not need to traverse from the vertices in \mathcal{IC} .

Proof According to Algorithm 4, since the vertices in \mathcal{IC} are in $\widehat{C}_k(\lambda_0)$ and $\widehat{C}_k(\{a\})$, for each $u \in \mathcal{IC}$, the value of $l_{\lambda_0}(u)$ and $l_a(u)$ are both $+\infty$. Thus, based on Definition 4, no vertex can form an active-path from the vertex in the \mathcal{IC} . Therefore, we do not need to traverse from the vertices in \mathcal{IC} . \square

Theorem 6 In the core-spread framework, for each vertex u in \mathcal{L} , we only need to traverse from its neighbors v in $V(\{a\})$ with higher layer value i.e., $l_a(v) > l_a(u)$; for each vertex u in \mathcal{R} , we only need to traverse from its neighbors v in $V(\lambda_0)$ with higher layer value i.e., $l_{\lambda_0}(v) > l_{\lambda_0}(u)$.

Proof Same as the proof for Theorem 5. According to Algorithm 4, since the vertices in \mathcal{L} are in $\widehat{C}_k(\lambda_0)$, the layer value of each vertex $u \in \mathcal{L}$ is set as $+\infty$ i.e., $l_{\lambda_0}(u) = +\infty$. Hence, the layer value of u 's neighbor in $V(\lambda_0)$ will not be larger than that of u , which means that there is no active-path from u to its neighbors. Besides, if u 's neighbors v in $V(\{a\})$ with no larger layer value i.e., $l_a(v) \leq l_a(u)$, they also can not form active-paths from u to v . Thus, we only need to traverse from u 's neighbors v in $V(\{a\})$ with higher layer value i.e., $l_a(v) > l_a(u)$. The proof for the vertices in \mathcal{R} is similar to that for \mathcal{L} . \square

Based on the above analysis, we can leverage the theorem to mark the vertices that can contribute to the degree of attribute k -core first. Generally, we mark the vertices along the active-path and the unmarked vertices definitely cannot exist in the attribute k -core (i.e., Algorithm 5). Then, we compute the newly degree upper bound of vertices based on the layer structure (i.e., Algorithm 6). The details of two algorithms are shown as follows.

Algorithm 5 MARK(λ_0, a).

Input : λ_0 : the currently selected attribute set, a : newly added attribute

Output : the number of marked vertices in \mathcal{UV}

```

1  $\mathcal{M} \leftarrow \mathcal{IV} \setminus \mathcal{UC}$ ;
2  $markSize \leftarrow |\mathcal{UC}|$ ;
3 foreach  $u \in \mathcal{UV}$  do
4    $m(u) \leftarrow 0$ ;
5 foreach  $u \in \mathcal{UC} \cup \mathcal{M}$  do
6    $m(u) \leftarrow 1$ ;
7 foreach  $u \in \mathcal{L}$  do
8   foreach  $v \in \widehat{N}(u, \{a\})$  and  $m(v) = 0$  do
9     if  $l_a(v) > l_a(u)$  then
10     $\mathcal{M}.push(v), m(v) \leftarrow 1$ ;
11 do the same process as Lines 7-10 by replacing  $\mathcal{L}$  with  $\mathcal{R}$  and  $\{a\}$  with  $\lambda_0$ ;
12 while  $\mathcal{M} \neq \emptyset$  do
13    $u \leftarrow \mathcal{M}.pop()$ ;
14    $markSize \leftarrow markSize + 1$ ;
15   foreach  $v \in \widehat{N}(u, \lambda_0 \cup \{a\})$  and  $m(v) = 0$  do
16     if  $v \in V(\lambda_0)$  and  $l_{\lambda_0}(v) > l_{\lambda_0}(u)$  then
17        $\mathcal{M}.push(v), m(v) \leftarrow 1$ ;
18     if  $v \in V(\{a\})$  and  $l_a(v) > l_a(u)$  then
19        $\mathcal{M}.push(v), m(v) \leftarrow 1$ ;
20 return  $markSize$ 

```

Mark Algorithm In the MARK algorithm, we use $m(u)$ to store u 's mark value and return the number of marked vertices. It starts by initializing \mathcal{M} with $\mathcal{IV} \setminus \mathcal{UC}$ and $markSize$ with the number of vertices in \mathcal{UC} in Lines 1-2. We set $m(u) = 0$ for each vertex $u \in \mathcal{UV}$ (Lines 3-4) and setting $m(u) = 1$ for each vertex $u \in \mathcal{UC}$ (Lines 5-6). Then, we try to visit each vertex in \mathcal{L} and check its neighbors in $V(\{a\})$ (Lines 7-10). According to Theorem 6, we do not need to start the traversal from neighbors in $V(\lambda_0)$. For each processed vertex u , if its resonate neighbor is not marked, we compare layer values of u and its neighbors. Note that, we use $l_a(u)$ to denote the layer value of u based on attribute set $\{a\}$. Specifically, if u

and its neighbor v satisfy $l_a(v) > l_a(u)$, which means v 's layer value is higher than u 's, we enlarge \mathcal{M} by pushing v and set $m(v) = 1$ (Lines 9-10). The same procedure as Lines 7-10 is conducted by replacing \mathcal{L} with \mathcal{R} and $\{a\}$ with λ_0 . We iteratively process vertices in \mathcal{M} in Lines 12-19.

For each processed vertex u , we visit its unmarked neighbors. If its neighbor is in $V(\lambda_0)$ and has larger layer value than u , we push this neighbor into \mathcal{M} and set it marked. Similarly, we push the neighbor of vertex u if its neighbor has larger layer value than u in $V(\{a\})$. Finally, we return the result in Line 20.

Algorithm 6: UPPER DEGREE(u).

```

Input :  $\lambda_0$  : the currently selected attribute set,  $a$  : newly added attribute,  $u$  : the
         checked vertex
Output : upper bound degree of  $u$  after updating
1  $nd^+(u) \leftarrow 0$ ;
2 foreach  $v \in \widehat{N}(u, \lambda_0 \cup \{a\})$  do
3   if  $v \in \mathcal{UC}$  or  $s(v) = 1$  then
4      $nd^+(u) \leftarrow nd^+(u) + 1$ ;
5   else if  $e(v) = 0$  and  $m(v) = 1$  then
6      $nd^+(u) \leftarrow nd^+(u) + 1$ ;
7 return  $nd^+(u)$ 
    
```

Upper Degree Algorithm Based on the layer structure and the MARK algorithm, we present the new upper bound degree $nd^+(u)$ for a vertex u . $nd^+(u)$ is the number of vertices in the union of u 's survived neighbors, u 's neighbors in \mathcal{UC} , u 's unexplored but marked neighbors. Details about computing the new upper bound degree for the chosen vertex u are shown in Algorithm 6. We initialize $nd^+(u)$ with 0 in Line 1 and process all neighbors of vertex u in Lines 2-6. For a vertex u , it will be removed if $nd^+(u) < k$, which can be verified by the following theorem.

Theorem 7 A vertex $u \in \mathcal{UV} \setminus \mathcal{UC}$ can not join in the updated attribute k -core if $nd^+(u) < k$.

Proof The neighbors of u can be divided into two categories, marked neighbors and unmarked neighbors. According to the properties of the Algorithm 5, unmarked neighbors

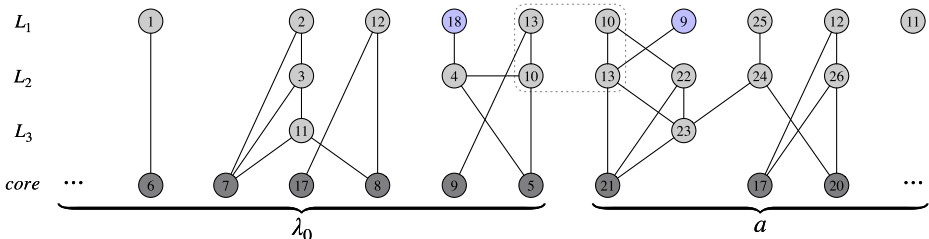


Figure 4 Example of the layer structure

will not be added to the updated attribute k -core. All marked neighbors may be counted into the upper degree of u except for the marked but deleted ones, and the deleted neighbors will definitely not be added to the updated attribute k -core. Thus, $nd^+(u)$ is a correct upper bound of u 's degree. If $nd^+(u) < k$, u can not be included into the attribute k -core. \square

Algorithm 7 COMPUTE ATTRIBUTE CORE(λ_0, a).

Input : G : an attribute graph, k : resonate degree constraint, λ_0 : the currently selected attribute set, a : newly added attribute

Output : $\widehat{C}_k(\lambda_0 \cup \{a\})$: the updated attribute k -core after adding a

- 1 $\mathcal{H}_a, \mathcal{H}_{\lambda_0} \leftarrow \emptyset$;
- 2 **foreach** $u \in \mathcal{UV}$ **do**
- 3 $e(u) \leftarrow 0, s(u) \leftarrow 0$;
- 4 **foreach** $u \in \mathcal{UC}$ **do**
- 5 $e(u) \leftarrow 1, s(u) \leftarrow 1, nd^+(u) \leftarrow +\infty$;
- 6 **foreach** $u \in \mathcal{L}$ **do**
- 7 **foreach** $v \in \widehat{N}(u\lambda_0 \cup \{a\})$ **do**
- 8 **if** $v \notin \mathcal{H}_a \cup \mathcal{H}_{\lambda_0} \cup \mathcal{UC}$ and $l_a(v) > l_a(u)$ **then**
- 9 $\mathcal{H}_a.push(v)$;
- 10 **do the same process as Lines 6-9 by replacing \mathcal{L} with \mathcal{R} and $\{a\}$ with λ_0 ;**
- 11 $\mathcal{H}_a \leftarrow \mathcal{IV} \setminus (\mathcal{IC} \cup \mathcal{R} \cup \mathcal{L}) \cup \mathcal{H}_a$; /* Theorem 5 */;
- 12 **while** $\mathcal{H}_a \cup \mathcal{H}_{\lambda_0} \neq \emptyset$ **do**
- 13 **while** $\mathcal{H}_a \neq \emptyset$ **do**
- 14 $u \leftarrow \mathcal{H}_a.pop()$;
- 15 $e(u) \leftarrow 1$;
- 16 $nd^+(u) \leftarrow \text{UPPERDEGREE}(u)$; /* Algorithm 6 */;
- 17 **if** $nd^+(u) \geq k$ **then**
- 18 $s(u) \leftarrow 1$;
- 19 **foreach** $v \in \widehat{N}(u\lambda_0 \cup \{a\})$ and $v \notin \mathcal{H}_a \cup \mathcal{H}_{\lambda_0}$ and $e(v) = 0$ **do**
- 20 **if** $u \in \mathcal{IV}$ **then**
- 21 **if** $l_a(v) > l_a(u)$ **then**
- 22 $\mathcal{H}_a.push(v)$;
- 23 **else if** $l_{\lambda_0}(v) > l_{\lambda_0}(u)$ **then**
- 24 $\mathcal{H}_{\lambda_0}.push(v)$;
- 25 **else if** $u \in V(\{a\})$ **then**
- 26 do the same process as Lines 21-22;
- 27 **else**
- 28 $s(u) \leftarrow 0$;
- 29 SHRINK(u); /* Algorithm 3 */;
- 30 **while** $\mathcal{H}_{\lambda_0} \neq \emptyset$ **do**
- 31 do the same process as Lines 14-29 by replacing $\{a\}$ with λ_0 ;
- 32 **using SHRINK algorithm to process all marked but unexplored vertices;**
- 33 **return** survived vertices

Table 2 Statistics of datasets

Dataset	Vertices	Edges	λ_{avg}	d_{avg}
Brightkite	58,228	214,078	11.48	7.35
Gowalla	196,591	950,327	8.37	9.67
Yelp	252,898	956,021	7.35	7.56
Youtube	1,157,828	2,987,625	9.51	5.16
DBLP	977,288	3,432,273	11.8	7.02
Flickr	581,099	4,972,275	9.90	17.11

4.3 Compute attribute core

Based on the new upper bound degree derived, Algorithm 7 is presented to incrementally compute the attribute k -core when adding a new attribute. We first initialize two heaps \mathcal{H}_a and \mathcal{H}_{λ_0} in Line 1, which indicate the unexplored vertices of $V(\{a\})$ and $V(\lambda_0)$, respectively. Then we set the explored value $e(u)$ and the survived value $s(u)$ of each vertex $u \in \mathcal{UV}$ as 0 in Lines 2-3. The vertices in \mathcal{UC} belong to the attribute k -core, so we set their explored value and survived value both as 1 and their upper bound degree as $+\infty$ (Lines 4-5). For each vertex $u \in \mathcal{L}$, we process its neighbors in Lines 7-9. If its neighbor v is not in $\mathcal{H}_a \cup \mathcal{H}_{\lambda_0} \cup \mathcal{UC}$ and has larger layer value than u , we push it into \mathcal{H}_a (Lines 8-9). We do the same process as Lines 6-9 by replacing \mathcal{L} with \mathcal{R} and $\{a\}$ with λ_0 . Then, we push all the vertices in $\mathcal{IV} \setminus (\mathcal{IC} \cup \mathcal{R} \cup \mathcal{L})$ (i.e., partition 3 discussed in Section 4.2) into \mathcal{H}_a for processing (Line 11). We iteratively process each vertex in $\mathcal{H}_a \cup \mathcal{H}_{\lambda_0}$ in Lines 12-31. If \mathcal{H}_a is not empty, for each vertex $u \in \mathcal{H}_a$, we first set its explored value $e(u)$ as 1 and compute its upper bound degree by invoking UPPER DEGREE(u) (Lines 14-16). u is survived if its upper degree is no less than k (Lines 17-18). The change of u 's state will effect its neighbors, so we process its unexplored neighbors iteratively in Lines 19-26. If $u \in \mathcal{IV}$ and $l_a(v) > l_a(u)$, we push v into \mathcal{H}_a (Lines 20-22). Similarly, if $u \in \mathcal{UV}$ and $\lambda_0(v) > \lambda_0(u)$, we push v into \mathcal{H}_{λ_0} (Lines 23-24). If $u \in V(\{a\})$, we do the same process as Lines 21-22.

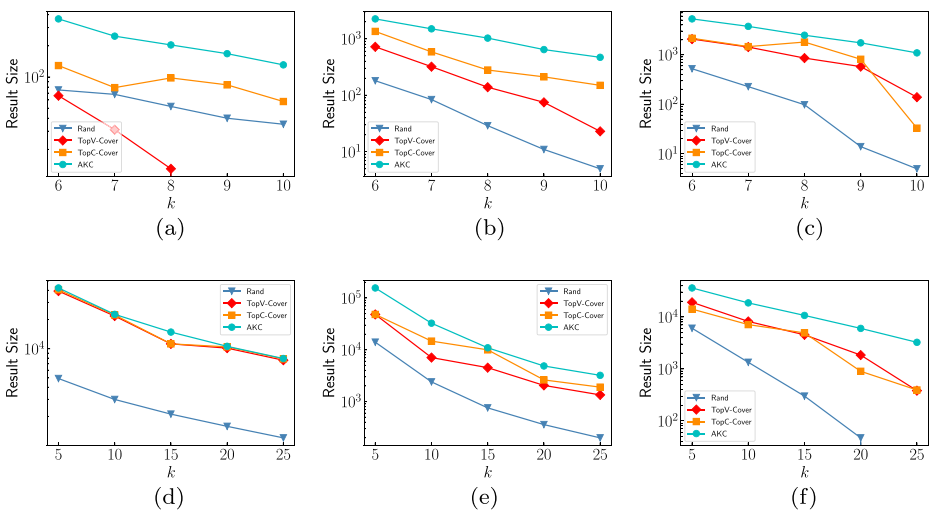


Figure 5 Effectiveness evaluation by varying k

u is deleted if $nd^+(u)$ is unsatisfied, and we invoke $\text{SHRINK}(u)$. If \mathcal{H}_{λ_0} is not empty, we do the same process as Lines 14–29 by replacing a and λ_0 . We stop the iteration when there is no vertex in $\mathcal{H}_a \cup \mathcal{H}_{\lambda_0}$. Then we invoke the SHRINK algorithm to process all marked but unexplored vertices because these vertices must be deleted. Finally, we return all survived vertices as the updated attribute k -core in Line 33.

Algorithm 8 AKC Algorithm.

Input : G : an attribute graph, k : resonate degree constraint, b : number of selected attributes

Output : A^* : the optimal b attributes

- 1 $G \leftarrow$ compute k -core in G ;
- 2 $\lambda_0 \leftarrow \emptyset$;
- 3 **foreach** $a \in \lambda$ **do**
- 4 $L_{\{a\}}, \widehat{C}_k(\lambda)\{a\} \leftarrow$ LAYER CONSTRUCTION($G, \{a\}, k$); /* Algorithm 4 */;
- 5 $\lambda_0 \leftarrow$ arg max $|\widehat{C}_k(\lambda)\{a\}|$;
- 6 **for** i from 2 to b **do**
- 7 $\delta \leftarrow -\infty$;
- 8 **foreach** $a \in A \setminus \lambda_0$ **do**
- 9 $UB(\lambda_0, a) \leftarrow$ MARK(λ_0, a); /* Algorithm 5 */;
- 10 **if** $UB(\lambda_0, a) < \delta$ **then** continue ;
- 11 $\widehat{C}_k(\lambda)\lambda_0 \cup \{a\} \leftarrow$ COMPUTE ATTRIBUTE CORE(λ_0, a); /* Algorithm 7 */;
- 12 **if** $|\widehat{C}_k(\lambda)\lambda_0 \cup \{a\}| > \delta$ **then**
- 13 $\delta \leftarrow |\widehat{C}_k(\lambda)\lambda_0 \cup \{a\}|$;
- 14 $a^* \leftarrow a$;
- 15 $\lambda_0 \leftarrow \lambda_0 \cup \{a^*\}$;
- 16 $L_{\lambda_0}, \widehat{C}_k(\lambda)\lambda_0 \leftarrow$ LAYER CONSTRUCTION(G, λ_0, k); /* Algorithm 4 */;
- 17 $A^* \leftarrow \lambda_0$;
- 18 **return** A^*

4.4 AKC algorithm

Suppose the currently selected attributes set is λ_0 and the newly added attribute is a . We use $UB(\lambda_0, a)$ to denote the upper bound size of updated attribute k -core, which can be obtained by the MARK algorithm, i.e., *markSize*. Specifically, there is no active-path contains the unmarked vertices, so the unmarked vertices cannot be added to the attribute k -core. Then, we have Theorem 8, which can further filter some unpromising attribute in current iteration. The correctness of the theorem is easy to verify. Thus, we omit the proof here.

Theorem 8 *In each iteration, if $UB(\lambda_0, a)$ is smaller than the current best result, attribute a can be filtered directly.*

By integrating all the techniques proposed, we come up with the optimized algorithm AKC, which details are shown in Algorithm 8. We obtain the k -core of G in Line 1 and initialize λ_0 as \emptyset to store the current best attribute set in Line 2. In Lines 3–4, we compute the layer structure and the corresponding attribute k -core of each attribute by invoking Algorithm 4. We put the best attribute with the largest attribute k -core into λ_0 (Line 5). In each

iteration, we initialize δ with $-\infty$ to denote the size of updated attribute k -core. For each unselected attribute $a \in A \setminus \lambda_0$, we compute the number of marked vertices by Algorithm 5 as the upper bound size of attribute k -core (Lines 8-9). This is because the unmarked vertices cannot be added into the attribute k -core. We continue the algorithm if $UB(\lambda_0, a) < \lambda$ (Line 10). Then, we compute the updated attribute k -core by Algorithm 7 (Line 11).

If the size of the current attribute k -core is larger than δ , we update δ and the best attribute a^* (Lines 12-14). The new current best attribute set λ_0 and layer information are updated in Lines 15 and 16. The algorithm terminates until b attributes are selected.

5 Experiments

In this section, we evaluate the effectiveness and efficiency of our proposed techniques on 6 real-word networks.

5.1 Experiment setup

Algorithms To the best of our knowledge, there is no existing work for our problem. For the effectiveness, we implement three algorithms (i.e., **Random**, **TopV-Cover** and **TopC-Cover**) to select different attribute set compared with our greedy strategy. We also implement and evaluate four algorithms (**Baseline**, **BL-S**, **BL-SA** and **AKC**) to verify the efficiency of proposed techniques. A brief description of the employed algorithms is as follows.

- **Exact**: the exact algorithm that enumerates all the combinations of attributes and returns the optimal result.
- **Random**: algorithm that randomly chooses b attributes from all attributes.
- **TopV-Cover**: algorithm that selects the top- b frequent attributes.
- **TopC-Cover**: algorithm that selects the top- b attributes with the largest corresponding attribute k -core size.
- **Baseline**: baseline greedy method, i.e., Algorithm 1.
- **BL-S**: greedy method which adopts the framework in Algorithm 2 to compute the attribute k -core.
- **BL-SA**: integrates BL-S with Theorems 5 and 6.
- **AKC**: algorithm that integrates all the developed techniques, i.e., Algorithm 8.

Datasets We conduct the experiments with 3 real-world attribute graphs and 3 semi-synthetic datasets. Table 2 presents the statistic details of the datasets, where Λ_{avg} is the average attribute number for each vertex and d_{avg} is the average degree of vertices. DBLP¹, Flickr² and Yelp³ are real-world attribute graphs. DBLP is an authors' relationship network with their paper information as keywords. For each author, we select the 30 most frequent keywords from all the titles of his published papers as his attributes. Flickr is an online user relationship network with their personal interests. For each user, we choose the 30 most frequent tags of its associated photos as his attributes. The Yelp dataset is extracted from the Yelp website, which consists of their businesses, reviews, and user data. The set of attributes

¹<http://dblp.uni-trier.de/xml>

²<https://www.flickr.com>

³<https://www.yelp.com/dataset>

for each user is the categories of the 30 most frequently visited restaurants by that user. The other 3 datasets, i.e., Brightkite, Gowalla and Youtube, are real-world networks, which are download from SNAP⁴. Since the vertices in these networks have no attributes, we first generate 400 attributes and then randomly select 20–30 attributes from 400 attributes as the attribute set for each vertex.

Parameters and workloads We conduct the experiments by varying the degree constraint k and the budget b . For parameter b , we vary it from 2 to 10 with $b = 6$ as the default value. Due to the different properties of the networks, for parameter k , we vary it from 5 to 25 with $k = 15$ as the default value for three real-world datasets, and vary it from 6 to 10 with $k = 8$ as the default value for the three semi-synthetic datasets. We evaluate the effectiveness and efficiency of the algorithms by reporting the identified the corresponding attribute k -core size and response time, respectively. For each setting, we run the algorithm 10 times and report the average value. All the programs are implemented in C++. The experiments are performed on a machine with an Intel i5-9600KF 3.7GHz CPU and 64 GB memory.

5.2 Effectiveness evaluation

To evaluate the effectiveness of proposed methods, we report the size of returned attribute k -core. Firstly, we conduct the experiments by comparing AKC with other 3 heuristic methods. Then, we report the results by comparing with the exact solution. Finally, we present the case studies on DBLP dataset.

Effectiveness evaluation by varying k and b Comparing AKC with Random, TopC-Cover and TopV-Cover, Figures 5 and 6 report the returned attribute k -core size by varying k and b , respectively. For Random, we report the average result by conducting 500 independent tests. As we can see, AKC always outperforms the other algorithms, and the two cover based methods are better than Random. The result size increases when b grows, because more attributes are selected. The core size decreases when k becomes larger, since vertices need larger degree to stay engaged. The result size of Random always very small due to the cardinality and distribution of Δ . Thus, Random may select a lot of unpromising attributes and lead to a smaller attribute k -core. Although the two cover based methods contain a lot of promising attributes, they do not consider the correlation among attributes. Thus, they still cannot perform as well as AKC.

Compare with the exact solution To further evaluate the effectiveness of proposed greedy framework, we report the results by comparing AKC with Exact. The experiments are conducted on two real-world datasets, i.e., DBLP and Flickr, and the results are shown in Figure 7. Due to the high computation cost of the exact solution, we only report the results for $b = 2$. The number on each bar denotes the corresponding running time. As shown, AKC achieves almost the same results as Exact. In addition, AKC is much faster than Exact. For example, on DBLP datasets with $k = 25$, AKC can finish in 0.81s, while it takes 63945s for the exact solution. Thus, the greedy framework can greatly accelerate the search with competitive results.

⁴<https://snap.stanford.edu/data>

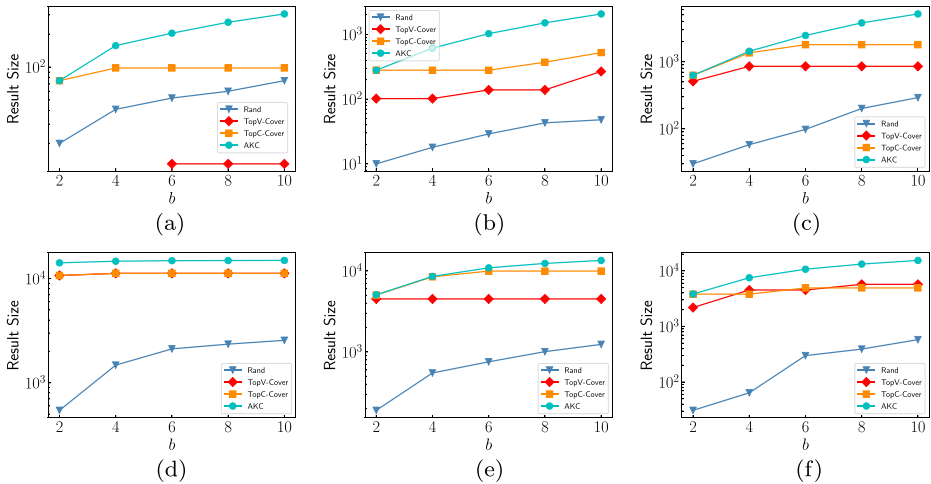


Figure 6 Effectiveness evaluation by varying b

Case Studies To demonstrate the properties of the investigated model, we conduct case studies on DBLP datasets, which results are shown in Figures 8 and 9. In Figure 8, with $k = 20$, the whole graph itself is a k -core. The red part is the attribute k -core depending on attribute set {language, structure, parallel, system, matrix}, while the blue vertex part is the attribute k -core formed by attribute set {graph, queue, network, complexity, search}. The number of vertices in red is 25, and the number of blue vertices is 43. Obviously, when different sets of attributes are selected, the corresponding attribute k -core is different and reveals different natures of the network. Thus, it is necessary to investigate the properties of attribute k -core. In Figure 9, for $b = 5$, the figure shows the corresponding attribute k -core returned by AKC when $k = 20$ and 30, respectively. In Figure 9a, the selected attribute set is {scheme, community, effective, technology, access}. In Figure 9b, the selected attribute set is {hoc, community, effective, technology, access}. For the convenience of viewing, we

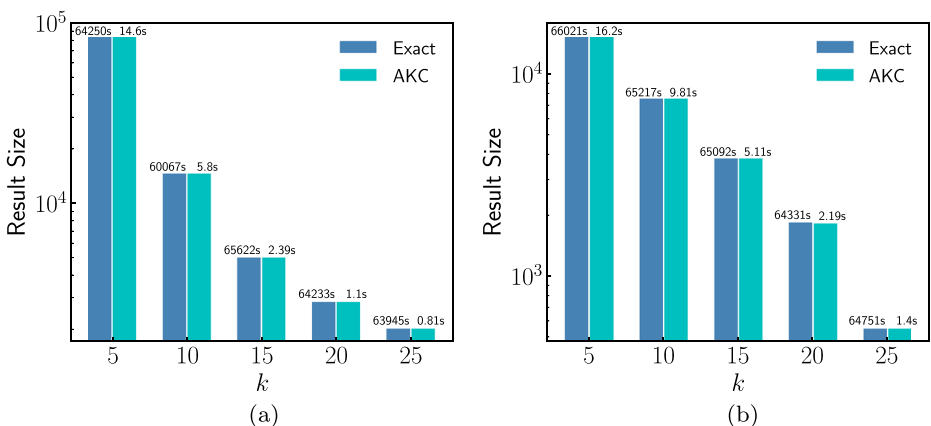


Figure 7 Compare with the exact solution

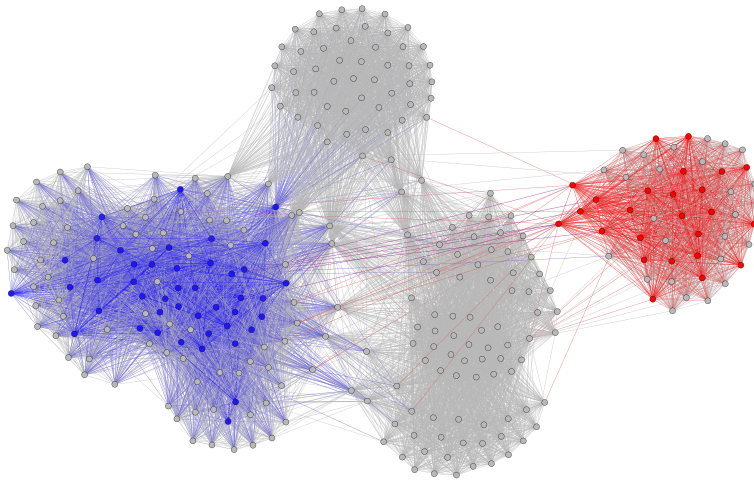


Figure 8 Case Studies on DBLP with $k = 20$

only draw the main component of each result. As observed, with the increase of k , the theme of the community also varies.

5.3 Efficiency evaluation

To evaluate the efficiency of proposed techniques, in this section, we conduct the experiments by comparing AKC with Baseline, BL-S and BL-SA.

Efficiency evaluation by varying k and b Figures 10 and 11 present the results by varying k and b , respectively. As we can see, AKC is much faster than the other three algorithms under all the settings. When increasing k , the response time is decreasing in most cases, since the k -core size decreases correspondingly. With the increase of b , the response time increases, because we need to perform more iterations to select sufficient attributes. As shown, BL-S has better performance compared to the Baseline, because BL-S only needs to conduct the computation for the vertices related to the currently accessed attribute set.

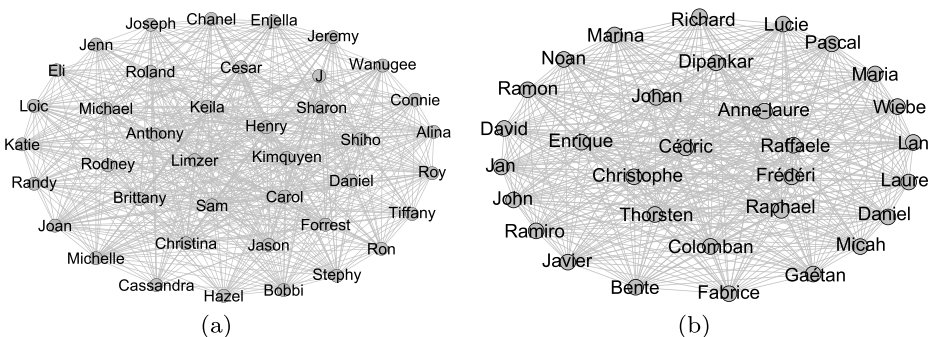


Figure 9 Case Studies on DBLP with $b = 5$

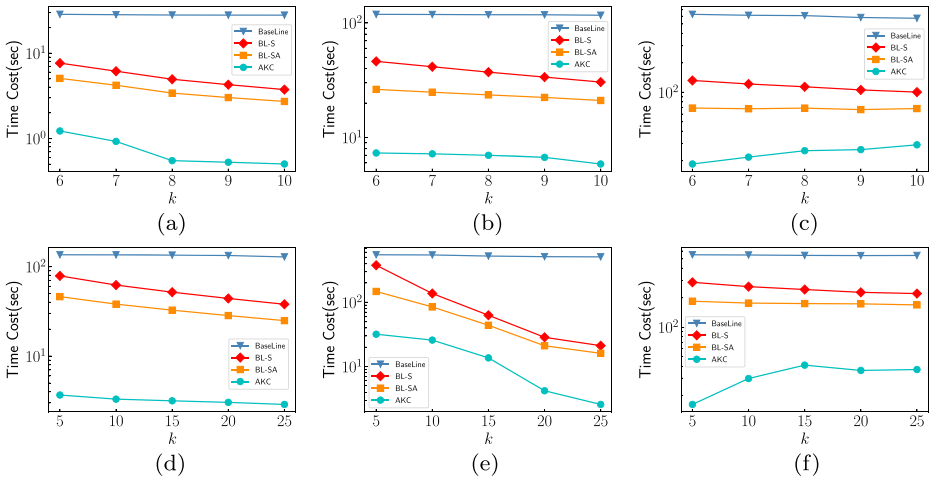


Figure 10 Efficiency evaluation by varying k

Although BL-SA only chunks the vertices in \mathcal{LV} compared to BL-S, it can skip many unnecessary explorations. Thus, BL-SA is faster than BL-S. As observed, with more technique equipped, the algorithm runs faster, which verifies the advantages of developed techniques.

Scalability evaluation In Figure 12, we evaluate the scalability of proposed methods. Specifically, we generate four subgraphs by randomly sampling 20-100% of the graph, and report the response time. Obviously, as the graph size grows, the response time increases. As observed, AKC is always faster than others, and preserve good scalability.

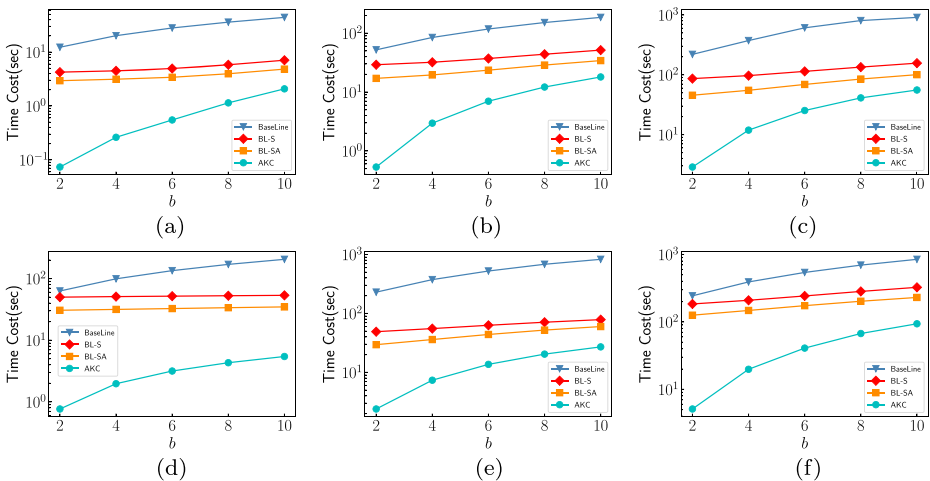


Figure 11 Efficiency evaluation by varying b

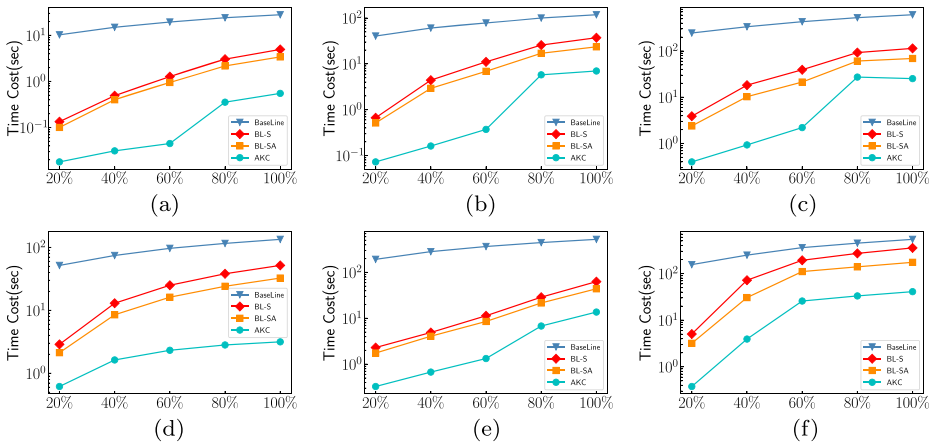


Figure 12 Scalability evaluation on all datasets

6 Related work

Cohesive subgraph detection is a fundamental problem in graph analysis. Different cohesive subgraph models are proposed in the literature, such as k -core [1], k -truss [19], clique [14], etc. As a popular model, k -core is widely adopted in many applications, e.g., community detection [21], influential community search [8], information propagation [2], etc. The k -core model is firstly introduced by Seidman in [12] for simple graphs. As the advance of online social network, users are often equipped with multiple attributes. In [20], Zhou et al. investigate graph clustering on attribute graph by considering both graph structural and attribute information. In [10], authors investigate the graph modeling with correlated attributes. [7] explores the attribute associations in the graphs. In [17], Yang et al. investigate the community detection problem in attribute graph. In [3] and [5], given a set of query attributes and query vertices, authors investigate the community search problem in attribute graph by leveraging the k -core and k -truss, respectively. In [9], a new community search model is further developed. As observed, in the literature, most of the researches focus on community search for a set of query attributes instead of identifying the critical attributes in the network. In addition, in their models, they usually emphasize each vertex containing one or all the query attributes instead of requiring the share of common attributes between neighbors. To the best of our knowledge, we are the first to investigate the attribute k -core maximization problem.

7 Conclusion

Identifying critical attributes is of great importance for attribute graph analysis. In this paper, we conduct the first research to investigate the attribute graph maximization problem. Given an attribute graph, it aims to retrieve a set of b attributes, which can lead to the largest attribute k -core. Due to the NP-hardness of the problem, a greedy framework is proposed. Layer-based filtering methods and searching paradigms are developed to scale for large

networks. Finally, experiments over real-life networks are conducted to demonstrate the effectiveness and efficiency of proposed model and techniques.

Acknowledgments This work is support by NSFC 61802345, ZJNSF LQ20F020007, ZJNSF LY21F020012 and Y202045024.

References

1. Batagelj, V., Zaversnik, M.: An $o(m)$ algorithm for cores decomposition of networks. arXiv:cs/0310049 (2003)
2. Caliò, A., Tagarelli, A., Bonchi, F.: Cores matter? an analysis of graph decomposition effects on influence maximization problems. In: ACM Conference on Web Science, pp. 184–193 (2020)
3. Fang, Y., Cheng, R., Luo, S., Hu, J.: Effective community search for large attributed graphs. VLDB (2016)
4. Guan, S., Ma, H., Wu, Y.: Attribute-Driven Backbone Discovery. In: KDD, pp. 187–195 (2019)
5. Huang, X., Lakshmanan, L.V.S.: Attribute-driven community search. PVLDB (2017)
6. Karp, R.M.: Reducibility among combinatorial problems. In: Complexity of Computer Computations, pp. 85–103. Springer (1972)
7. Lee, J., Park, K., Prabhakar, S.: Mining statistically significant attribute associations in attributed graphs. In: ICDM, pp. 991–996 (2016)
8. Li, R.H., Qin, L., Yu, J.X., Mao, R.: Influential community search in large networks. PVLDB **8**(5), 509–520 (2015)
9. Liu, Q., Zhu, Y., Zhao, M., Huang, X., Xu, J., Gao, Y.: Vac: Vertex-Centric Attributed Community Search. In: ICDE, pp. 937–948 (2020)
10. Pfeiffer, J.J. III., Moreno, S., La Fond, T., Neville, J., Gallagher, B.: Attributed graph models: Modeling network structure with correlated attributes. In: WWW, pp. 831–842 (2014)
11. Qi, G.J., Aggarwal, C.C., Huang, T.: Community detection with edge content in social media networks. In: ICDE, pp. 534–545 (2012)
12. Seidman, S.B.: Network structure and minimum degree. Social Networks (1983)
13. Sun, R., Chen, C., Wang, X., Zhang, Y., Wang, X.: Stable community detection in signed social networks. TKDE (2020)
14. Sun, R., Zhu, Q., Chen, C., Wang, X., Zhang, Y., Wang, X.: Discovering cliques in signed networks based on balance theory. In: DASFAA, pp. 666–674 (2020)
15. Wang, X., Zhang, Y., Zhang, W., Lin, X.: Efficient distance-aware influence maximization in geo-social networks. TKDE **29**(3), 599–612 (2016)
16. Wang, X., Zhang, Y., Zhang, W., Lin, X., Chen, C.: Bring order into the samples: A novel scalable method for influence maximization. TKDE **29**(2), 243–256 (2016)
17. Yang, J., McAuley, J., Leskovec, J.: Community Detection in Networks with Node Attributes. In: ICDM, pp. 1151–1156 (2013)
18. Zhang, F., Zhang, Y., Qin, L., Zhang, W., Lin, X.: When engagement meets similarity: efficient (k, r) -core computation on social networks. PVLDB **10**(10), 998–1009 (2017)
19. Zhao, J., Sun, R., Zhu, Q., Wang, X., Chen, C.: Community Identification in Signed Networks: a K-Truss Based Model. In: CIKM, pp. 2321–2324 (2020)
20. Zhou, Y., Cheng, H., Yu, J.X.: Graph clustering based on structural/attribute similarities. VLDB (2009)
21. Zhu, R., Zou, Z., Li, J.: Fast diversified coherent core search on multi-layer graphs. VLDB J. **28**(4), 597–622 (2019)
22. Zhu, W., Zhang, M., Chen, C., Wang, X., Zhang, F., Lin, X.: Pivotal relationship identification: The K-Truss minimization problem. In: IJCAI, pp. 4874–4880 (2019)

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.