J Supercomput
DOI 10.1007/s11227-017-1976-z

An attribute-based community search method with graph refining

Jingwen Shang[1] · **Chaokun Wang**[1] · **Changping Wang**[1] · **Gaoyang Guo**[1] · **Jun Qian**[1]

Abstract In many complex networks, there exist diverse network topologies as well as node attributes. However, the state-of-the-art community search methods which aim to find out communities containing the query nodes only consider the network topology, but ignore the effect of node attributes. This may lead to the inaccuracy of the predicted communities. In this paper, we propose an attribute-based community search method with graph refining technique, called AGAR. First, we present the concepts of topology-based similarity and attribute-based similarity to construct a TA-graph. The TA-graph can reflect both the relations between nodes from the respect of the network topology and that of the node attributes. Then, we construct AttrTCP-index based on the structure of TA-graph. Finally, by querying the AttrTCP-index, we can find out the communities for the query nodes. Experimental results on real-world networks demonstrate AGAR is an effective and efficient community search method by considering both the network topology and node attributes.

Keywords Community search · Graph refining · Node attributes · Network topology

✉ Chaokun Wang
chaokun@tsinghua.edu.cn

Jingwen Shang
shangjw15@mails.tsinghua.edu.cn

Changping Wang
wang-cp12@mails.tsinghua.edu.cn

Gaoyang Guo
ggy16@mails.tsinghua.edu.cn

Jun Qian
qian-j16@mails.tsinghua.edu.cn

[1] School of Software, Tsinghua University, Beijing 100084, China

Published online: 13 February 2017

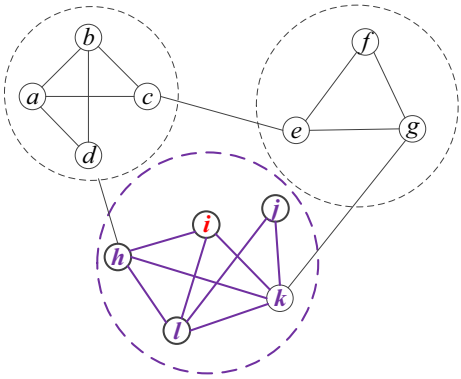


Fig. 1 A graph containing three communities

1 Introduction

The community structure is one of the most important characteristics of complex networks. Topologically, a community is a cohesive subgraph where nodes are densely connected, while nodes in different communities are sparsely connected. Generally, nodes in the same community tend to have common attributes or similar functions [12].

Community detection aims to find out all communities within an input network. It is an important research problem in the fields such as computer science, physics, biology and communication and has been studied a lot in last decades [12,23,25,28,36,39,40]. As an interesting counterpart, community search seeks to find the communities containing given objects, which has attracted much attention recently [8,31,32]. The main difference between these two tasks is that community detection targets all communities in the entire network according to a global criterion [12], while community search only focuses on the personalized communities for a query node or a group of query nodes [32].

In a network, as the communities for different nodes may have great differences, it is more meaningful to explore and study the user-centered personalized community search problem [16]. For example, by utilizing the general community detection method, the network in Fig. 1 can be divided into three communities surrounded by the dotted circles. However, in some applications, we are only interested in the node i and would like to know the communities containing it. In such a case, using the community detection method to find out all the communities and then finding the required community from them may be a waste of time. It is more efficient to directly adopt the community search method to predict the community containing the given node i, which consists of the nodes h, i, j, k and l.

In recent years, several community search methods have been brought forth [2,9,16,17,32]. However, the state-of-the-art community search methods only consider the influence of network topology, ignoring the effect of node attributes, which may lead to the inaccuracy of resulted communities.

Example 1 As shown in Fig. 2, a co-authorship network consists of nodes which represent authors of papers. If a couple of scholars co-author one paper, there exist

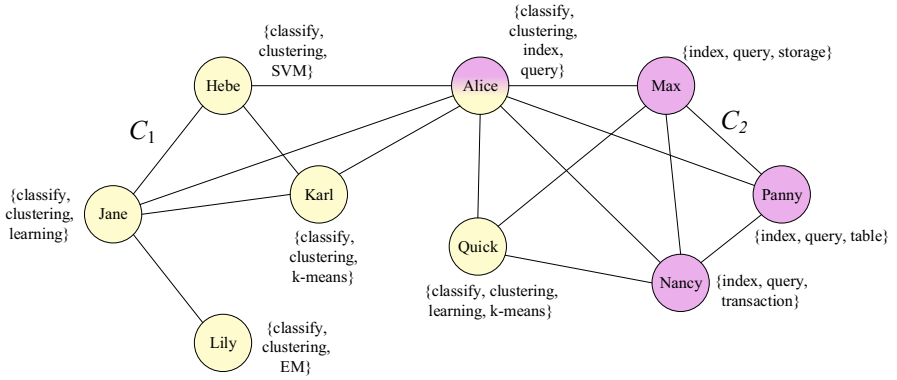


Fig. 2 A co-authorship network

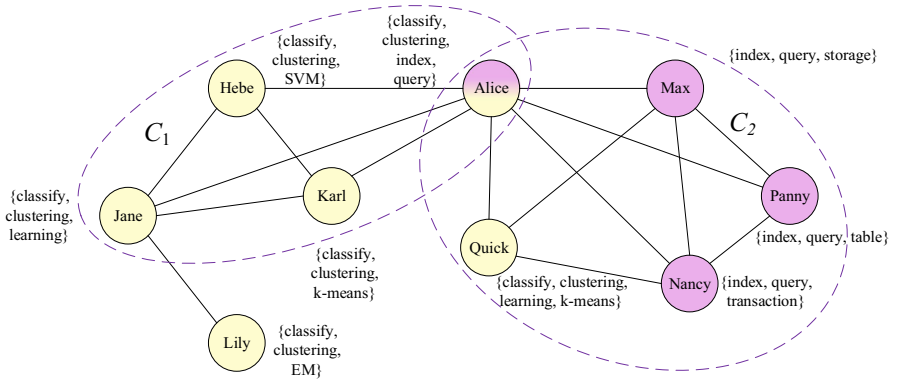


Fig. 3 Communities within the co-authorship network obtained by general community search methods

edges between the corresponding nodes. Besides, each node has a set of keywords which reveal the node attributes. The node color indicates the category (also called community) of the node. In detail, purple nodes (i.e., *Max*, *Nancy* and *Panny*) are in the category of database; yellow nodes (i.e., *Quick*, *Hebe*, *Jane*, *Karl* and *Lily*) are in the category of machine learning. The node *Alice* is in both the machine learning category and the database category.

By only considering the network topology and ignoring the node attributes, the general community search methods find out the communities containing the node *Alice*, as shown in Fig. 3. Based on the network topology, as the node *Lily* is not densely connected with other nodes, it cannot be assigned to community C_1. Besides, as the node *Quick* is densely connected with other nodes in C_2, it is assigned to community C_2. In fact, *Lily* should be assigned to C_1 because she belongs to the machine leaning category just as other nodes in C_1. Meanwhile, *Quick* should be assigned to C_1 instead of C_2, for he belongs to the machine learning category rather than the database category. Clearly, by using the general community search methods which only explore the network topology, it is difficult to find out the accurate communities for the given nodes.

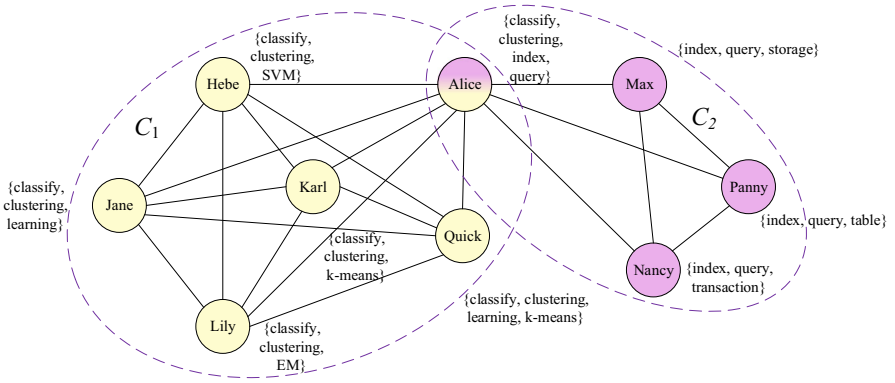


Fig. 4 Communities within the TA-graph derived from the co-authorship network obtained by our method

Through the above example, we can see that there are two major deficiencies in the state-of-the-art community search methods.

1. Only considering the influence of network topology structures ignores the effect of node attributes to communities.
2. Existing community search methods, especially the k-truss method, are more suitable to dense networks, but may cause inaccuracy for sparse networks.

In this paper, we propose an Attribute-based community search method with GrAph Refining, called AGAR. By revealing the relations between nodes from both the network topology and node attributes, AGAR can find out more accurate communities for the given nodes. Please note this paper is an extended version of our previous conference paper [31].

Example 2 Given the initial co-authorship network shown in Fig. 2, our method can strengthen the relations between nodes in the same categories and weaken the relations between nodes in different categories. Then, we can construct the TA-graph as shown in Fig. 4 according to the topology-based similarity and attribute-based similarity. Finally, we can solve the problem in Example 1 and generate the accurate communities for the node *Alice*.

In summary, the main contributions of this paper are as follows.

1. We construct a TA-graph structure which can reflect both the relations between nodes from the respect of the network topology and that of the node attributes by computing the topology-based similarity and the attribute-based similarity for the networks.
2. Based on the keywords which reveal the node attributes, we build the AttrTCP-index where each node has a set of keywords and a union of all of the keywords of its neighbors. For the given node, we can efficiently find the communities which satisfy the personalized requirements for the community scale.
3. Experimental results conducted on real-world networks demonstrate that AGAR can find out more accurate communities by capturing the information of both network topology and node attributes.

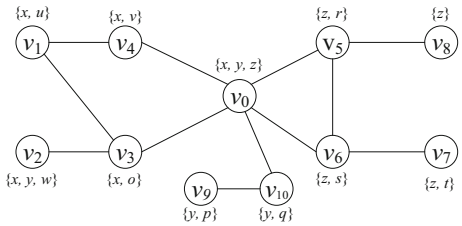


Fig. 5 A graph G

The rest of this paper is organized as follows. Several useful concepts as well as the formal description of the attribute-based community search problem are presented in Sect. 2. The community search method AGAR is proposed in Sect. 3. Extensive experimental results on real-world networks are reported in Sect. 4. The related work is discussed in Sect. 5. We conclude this paper in Sect. 6.

2 Preliminaries

In this section, some useful concepts and the problem description addressed in this paper are brought forward.

Let $G = (V, E, X)$ be an undirected attributed graph with $n = |V|$ nodes and $m = |E|$ edges. Different from the general graph, each node in the attributed graph has a set of keywords which represent the node attributes. X denotes the content matrix $X \in \{0, 1\}^{n \times l}$, where l is the number of keywords of all the nodes in G. The entry $x_{ij} = 1$ if the ith node has the jth keyword, and $x_{ij} = 0$ otherwise. We use $N(u)$ to represent the set of neighbors of a node u, and $d(u) = |N(u)|$. The symbol $\triangle_{uvw}$ denotes all the circles formed by three nodes u, v, w and the three edges between them.

The *topology-based similarity* reveals the similarity of nodes in G from the respect of topology. Generally, the topology-based similarity indicates that if two nodes have some common neighbors, they are likely to be similar. Based on this idea, we present the definition of topology-based similarity as follows. There are some other methods to compute the topology-based similarity as well [18,43].

Definition 1 (*Topology-based similarity*) Given an attributed graph $G = (V, E, X)$, the topology-based similarity between a pair of nodes in G is defined as

$$topo\text{-}sim(u, v) = \frac{2|N(u) \cap N(v)|}{d(u) + d(v)} \tag{1}$$

where $u, v \in V$.

Example 3 In the attributed graph G as shown in Fig. 5, the nodes v_3 and v_4 have two common neighbors (i.e., v_0 and v_1). Then, $topo\text{-}sim(v_3, v_4) = \frac{2|N(v_3) \cap N(v_4)|}{d(v_3)+d(v_4)} = \frac{2\times 2}{2+3} = 0.8$. The high value of $topo\text{-}sim(v_3, v_4)$ indicates that the nodes v_3 and v_4 are similar to each other and are likely to belong to the same community.

Given an attributed graph G, the *attribute-based similarity* represents the node similarity from the respect of node attributes. The attribute-based similarity indicates that if the nodes have many common keywords, they tend to be similar. In this study, we adopt the cosine similarity to quantitatively evaluate the attribute-based similarity for a pair of nodes in graph G [24]. Clearly, other proper metrics could be used as well. The definition of attribute-based similarity is as follows.

Definition 2 (*Attribute-based similarity*) Given an attributed graph $G = (V, E, X)$, $\forall v_i, v_j \in V$, the cosine similarity $sim(v_i, v_j)$ of the two nodes v_i and v_j, whose corresponding content vectors are x_i and x_j in X, is defined as

$$attr\text{-}sim(v_i, v_j) = \frac{x_i \cdot x_j^T}{||x_i||_2||x_j||_2} \tag{2}$$

Example 4 As shown in Fig. 5, the nodes v_0 and v_8 have the common keyword $\{z\}$. Then, $attr\text{-}sim(v_0, v_8) = \frac{x_0 \cdot x_8^T}{||x_0||_2 \times ||x_8||_2} = \frac{(1,1,1,0,0,0,0,0,0,0,0,0) \cdot (0,0,1,0,0,0,0,0,0,0,0,0)^T}{||(1,1,1,0,0,0,0,0,0,0,0,0)||_2 \times ||(0,0,1,0,0,0,0,0,0,0,0,0)||_2}$ $= 0.58$. The high value of $attr\text{-}sim(v_0, v_8)$ indicates that the nodes v_0 and v_8 are similar to each other from the node attribute respect and are likely to belong to the same community, although they are not adjacent and are not similar from the topology respect.

Definition 3 (*TA-graph*) Given an attributed graph $G = (V, E, X)$, the Topology- and Attribute-based graph, denoted as TA-graph, is the graph constructed by computing the topology-based similarity and attribute-based similarity. The TA-graph can reflect the relations between nodes from the respect of the topology and that of node attributes.

There are many methods to construct the TA-graph from different respects. In this work, we propose the graph refining algorithm that constructs the TA-graph by computing the topology-based similarity and attribute-based similarity.

The problem of attribute-based community search studied in this paper is presented as follows.

Problem definition Given an undirected attributed graph $G = (V, E, X)$, a query node $v_q \in V$, the parameter $k \geq 2$ controlling the density of nodes in topology and the parameter *queryNum* limiting the similarity of node attributes, the problem of attribute-based community search is to find all communities containing v_q, where the nodes satisfy the requirements from topology and node attributes.

3 AGAR

In this section, we propose the algorithms for AGAR, which can obtain more accurate communities by exploring both network topology and node attributes. First, we propose the graph refining algorithm to construct the TA-graph by computing the topology-based similarity and attribute-based similarity between nodes. Next, we compute the AttrSup of edges in the TA-graph and construct the AttrTCP-index based on

Algorithm 1 Graph Refining

Input: $G = (V, E), \alpha$
Output: $TA\text{-}graph = (TA\text{-}V, TA\text{-}E)$
1: $E_c = \emptyset$, $TA\text{-}graph = G$;
2: **for** $(\forall v_i, v_j \in V, i \neq j)$ **do**
3: Compute the cosine similarity value $attr\text{-}sim(x_i, x_j)$;
4: $E_c \leftarrow [(v_i, v_j), attr\text{-}sim(x_i, x_j)]$;
5: **end for**
6: $simR \leftarrow$ the rth largest similarity value in E_c;
7: $simL \leftarrow$ the lth smallest similarity value in E_c;
8: **for** $\forall e \in TA\text{-}E$ **do**
9: **if** $attr\text{-}sim(e) \leq simL$ **then**
10: remove e from $TA\text{-}E$;
11: **end if**
12: **end for**
13: **for** $\forall x \in V$ **do**
14: **for** $\forall y, z \in N(x)$ **do**
15: **if** $((y, z) \notin TA\text{-}E) \& (attr\text{-}sim(y, z) \geq simR \parallel topo\text{-}sim(y, z) \geq \alpha)$ **then**
16: $TA\text{-}E \leftarrow (y, z)$;
17: **end if**
18: **end for**
19: **end for**
20: return $TA\text{-}graph$.

k-truss method as well as the set of keywords for nodes. Finally, we search for the communities including the given node, which can satisfy the personalized requirements for the community scale.

3.1 Graph refining

In this section, we propose the graph refining algorithm for AGAR, the basic idea of which is as follows. Firstly, the TA-graph is initialized to the graph G. We compute the attribute-based similarity values of any pairs of nodes in the graph G. Next, the edges between nodes with low attribute-based similarity values are removed from the TA-graph. Then, the edges between nodes with high topology-based or attribute-based similarity values are added into the TA-graph. In this way, the TA-graph is constructed by computing the topology-based similarity and attribute-based similarity. After the graph refining process, the topology of TA-graph reveals more accurate similarity relations between nodes of the graph. Besides, the topology of the new TA-graph is more densely connected.

The graph refining process is outlined in Algorithm 1. After the initialization (Line 1), the attribute-based similarity values of all pairs of nodes are computed and inserted into the similarity set E_c (Lines 2–5). Then, let $simR$ be the rth largest similarity value in E_c, and $simL$ be the lth smallest similarity value, where $r = l = m$ (Lines 6–7). We set the values of r and l to be the number of edges m, which can balance the effect of network topology and node attributes on the structure of TA-graph. For each edge $e \in TA\text{-}E$, if the attribute-based similarity $attr\text{-}sim(e) \leq simL$, remove e from $TA\text{-}E$; otherwise, e remains unchanged (Lines 8–12). For each node $x \in V$,

Table 1 Topology-based similarity values of some pairs of nodes in G

Pairs of nodes	*topo-sim*
$(\mathbf{v_0}, \mathbf{v_1})$	**0.57**
(v_0, v_2)	0.33
(v_0, v_7)	0.33
(v_0, v_8)	0.33
(v_0, v_9)	0.33
$(\mathbf{v_1}, \mathbf{v_2})$	**0.67**
(v_2, v_4)	0
$(\mathbf{v_3}, \mathbf{v_4})$	**0.80**
$(\mathbf{v_5}, \mathbf{v_7})$	**0.50**
$(\mathbf{v_6}, \mathbf{v_8})$	**0.50**

Table 2 Attribute-based similarity values of some pairs of nodes in G

Pairs of nodes	*attr-sim*
(v_0, v_1)	0.41
$(\mathbf{v_0}, \mathbf{v_2})$	**0.67**
(v_0, v_7)	0.41
$(\mathbf{v_0}, \mathbf{v_8})$	**0.58**
(v_0, v_9)	0.41
(v_1, v_2)	0.41
(v_2, v_4)	0.41
$(\mathbf{v_3}, \mathbf{v_4})$	**0.50**
$(\mathbf{v_5}, \mathbf{v_7})$	**0.50**
$(\mathbf{v_6}, \mathbf{v_8})$	**0.50**

let y and z be a couple of neighbors of x which are not adjacent to each other. If $attr\text{-}sim(y, z) \geq simR$ or $topo\text{-}sim(y, z) \geq \alpha$, add edge (y, z) into the edge set $TA\text{-}E$ of $TA\text{-}graph$, where α is set to 0.5, which means the size of common neighbors is half of the average size of neighbors of nodes y, z (Lines 13–19). Finally, the TA-graph is returned (Line 20).

In Algorithm 1, the time complexity to compute E_c is $O(n^2)$. Then, the time complexity to obtain the rth largest and the lth smallest similarity value in E_c is $O(2n^2)$. All the edges in $TA\text{-}E$ are traversed to remove the edges which do not satisfy the requirements (Lines 8–12) in $O(m)$ time. Next, the time complexity to add edges to $TA\text{-}E$ (Lines 13–19) is $O(m^{1.5}d_{max})$, where d_{max} is the maximum of degrees of all nodes [30]. Thus, the time complexity of Algorithm 1 is $O(n^2 + m^{1.5}d_{max})$, and the space complexity is $O(n^2)$.

Example 5 For the graph shown in Fig. 5, the topology-based similarity values and the attribute-based similarity values of any pairs of nodes are computed, and part of the results are shown in Tables 1 and 2, respectively. In Table 1, we choose $\alpha = 0.5$. For the pairs of nodes (v_0, v_1), (v_1, v_2), (v_3, v_4), (v_5, v_7), (v_6, v_8), $topo\text{-}sim((u, v)) \geq \alpha$, which means they are similar in topology. In Table 2, we choose $r = l = m = 12$,

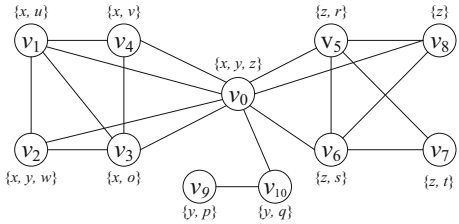


Fig. 6 The TA-graph of G

$simR = 0.5$, and $simL = 0$. For all the pairs of nodes in G, the attribute-based similarity values between nodes (v_0, v_2), (v_0, v_8), (v_3, v_4), (v_5, v_7), (v_6, v_8) satisfy that $attr\text{-}sim(u, v) \geq simR$. Thus, the corresponding edges are added into TA-graph. After the graph refining process, we can obtain the TA-graph as shown in Fig. 6.

3.2 Index construction

After the graph refining process, Algorithm 2 is proposed to build the AttrTCP-index for the nodes in the TA-graph.

First, the AttrSup of edge is computed according to the topology and keywords in TA-graph (Lines 1–20). The support of an edge e in a graph G is the number of triangles containing e. For example, in the graph in Fig. 1, the edge $e(h, i)$ is contained in triangles $\triangle_{hil}$ and $\triangle_{hik}$. Thus, $sup((h, i)) = 2$. Different from the definition of support, AttrSup is computed based on the updated TA-graph according to the topology and the node keywords. In the process to compute the AttrSup values for edges, new edges are added into the TA-graph to improve the structure of it. For each edge $e = (u, v)$, we check the common keywords between nodes u and $x \in N(v)$ (Line 8). The inequation $2|u.keywords \cap x.keywords|/(|u.keywords| + |x.keywords|) \geq \beta$ represents that if the number of common keywords of node u and x is a relatively large proportion of the average number of keywords of u and x, an edge between node u and x can be added into the TA-graph, where we set $\beta = 1/3$ (Lines 7–12). The nodes v and $x \in N(v)$ follow the same operation as u and $x \in N(v)$ (Lines 13–18). Then, we compute the values of AttrSup of all edges in the updated structure of TA-graph (Line 20). Next, we run the k-truss decomposition to compute the AttrTrussness values for edges in the improved TA-graph (Line 21) [35]. Then, the attributed triangle connectivity preserved index, denoted as AttrTCP-index, for each node in G is constructed based on the structure of TA-graph (Lines 22–26). Different from the TCP-index proposed in [16], in our AttrTCP-index, each node u contains its own set of keywords $u.keywords$ and the union of keywords of its neighbors $u.nei\_key\_union$. Finally, the AttrTCP-index is returned (Line 27).

In Algorithm 2, all the edges are pushed into Q (Lines 2–4) in $O(m)$ time. The computation of the AttrSup values of the edges can be done in $O(m^{1.5})$ [30]. Next, the time complexity to construct AttrTCP-index is $O(\sum_{(u,v)\in E} min\{d(u), d(v)\}) \subseteq O(m^{1.5})$ [16]. Thus, the time complexity of Algorithm 2 is $O(m^{1.5})$ and the space complexity is $O(m)$.

Algorithm 2 AttrTCP-index Construction

Input: $TA\text{-}G = (TA\text{-}V, TA\text{-}E), \beta$
Output: AttrTCP-index for each node
1: $Q = \emptyset$;
2: **for** $\forall e \in TA\text{-}E$ **do**
3: $Q.push(e)$;
4: **end for**
5: **while** $Q \neq \emptyset$ **do**
6: $e(u, v) = Q.pop()$;
7: **for** $\forall x \in N(v)$ **do**
8: **if** $((u, x) \notin TA\text{-}E \& 2|u.keywords \cap x.keywords|/(|u.keywords| + |x.keywords|) \geq \beta)$ **then**
9: $Q.add((u, x))$;
10: $TA\text{-}E.add((u, x))$;
11: **end if**
12: **end for**
13: **for** $\forall x \in N(u)$ **do**
14: **if** $((x, v) \notin TA\text{-}E \& 2|v.keywords \cap x.keywords|/(|v.keywords| + |x.keywords|) \geq \beta)$ **then**
15: $Q.add((x, v))$;
16: $TA\text{-}E.add((v, x))$;
17: **end if**
18: **end for**
19: **end while**
20: The value of AttrSup(e) for each edge e is computed according to the updated TA-graph.
21: Run k-truss decomposition to obtain the AttrTrussness value for each edge in TA-graph according to the AttrSup of edges;
22: **for** $k \leftarrow k_{max}$ to 2 **do**
23: Construct AttrTCP-index by adding edges (x, y) with different connected components;
24: $x.nei\_key\_union.add(y.keywords)$;
25: $y.nei\_key\_union.add(x.keywords)$;
26: **end for**
27: return AttrTCP-index;

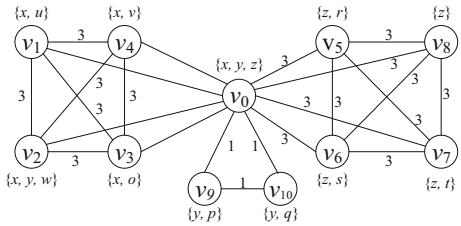


Fig. 7 The improved TA-graph after the process of computing the AttrSup for each edge

Example 6 For the TA-graph shown in Fig. 6, after the processing of computing the AttrSup values for edges, the improved structure of TA-graph with AttrSup values is shown in Fig. 7. Compared with the structure of TA-graph shown in Fig. 6, the improved TA-graph adds the edges for (v_2, v_4), (v_7, v_8), (v_0, v_7) and (v_0, v_9) as $2|v_2.keywords \cap v_4.keywords|/(|v_2.keywords| + |v_4.keywords|) = 2/5 \geq 1/3$, $2|v_7.keywords \cap v_8.keywords|/(|v_7.keywords| + |v_8.keywords|) = 2/3 \geq 1/3$, $2|v_0.keywords \cap v_7.keywords|/(|v_0.keywords| + |v_7.keywords|) = 2/5 \geq 1/3$, and $2|v_0.keywords \cap v_9.keywords|/(|v_0.keywords| + |v_9.keywords|) = 2/5 \geq 1/3$. After the index constructing process, we can obtain the AttrTCP-index with AttrTrussness values as shown in Fig. 8.

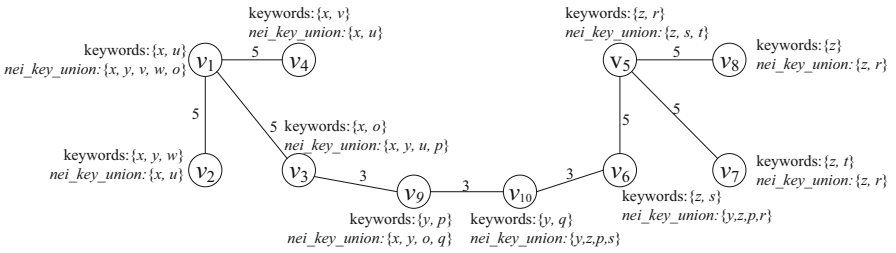


Fig. 8 The AttrTCP-index with AttrTrussness values for v_0 in G

3.3 Query processing

In this subsection, given the AttrTCP-index, k, *queryNum* and the query node v_q, we present the method to search the diverse scaled communities $C_1, \ldots, C_L$. Different from the querying process of the general k-truss methods [16], we can select the number of query keywords *queryNum*. Before adding the edge (x, y) into community C_l $(1 \leq l \leq L)$, we first determine whether the keywords of x and y satisfy the requirements $(|x.keywords \cap v_q.keywords| \geq queryNum) \& (|y.keywords \cap v_q.keywords| \geq queryNum)$. If the requirements above are satisfied, the edge (x, y) is added into community C_l. The *queryNum* requires that the number of common keywords for the query node v_q and other nodes in the communities should be no less than *queryNum*. By setting the value of *queryNum*, we can control the scale for the predicted communities. On the one hand, if we prefer to find larger communities which include all the related nodes with v_q, we can set *queryNum* to be a small number like 1. On the other hand, if we require that the nodes have very close relations on both topology and node attributes, we can set *queryNum* to be a larger number.

Besides, we utilize the union of keywords of u's neighbors nei_key_union to help the querying process. If the size of the intersection of v_q's keywords and the keywords of the u's neighbors is less than *queryNum*, there is no need to search the neighbors of node u. Before adding the reversed edge (v, u) of edge (u, v) into the searching queue of edges Q, we judge whether the size of keywords union of v satisfies the requirement $|v.nei\_key\_union \cap v_q.keywords| \geq queryNum$. The reversed edge (v, u) is added into Q when the requirement above is satisfied. By utilizing this pruning strategy, we can generate the communities more smoothly. The time complexity of the querying process is $O(|C_1 \cup \cdots \cup C_l|)$.

Example 7 For the AttrTCP-index for v_0 in Fig. 8, if we select $queryNum = 1, k = 3$, the large-scale communities containing v_0 after the querying operation are shown in Fig. 9. If we select $queryNum = 1, k = 5$, the medium-scale communities containing v_0 after the querying operation are shown in Fig. 10. If we select $queryNum = 2$, $k = 5$, the small-scale communities containing v_0 after the querying operation are shown in Fig. 11.

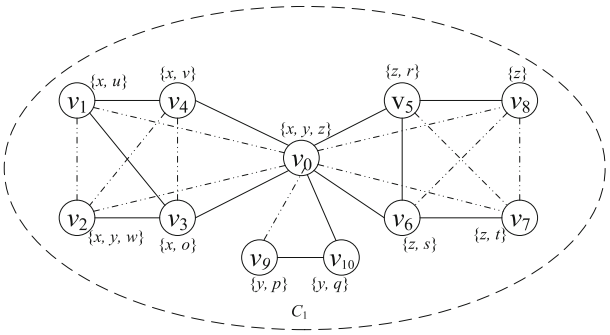


Fig. 9 The large-scale communities for v_0 in G

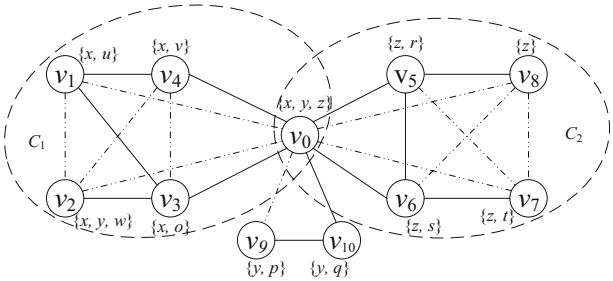
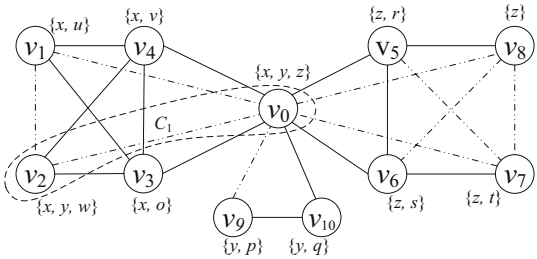


Fig. 10 The medium-scale communities for v_0 in G

Fig. 11 The small-scale communities for v_0 in G



4 Experiments

4.1 Experimental preparation

We evaluate efficiency and effectiveness of our proposed method AGAR on real-world networks. All algorithms are implemented in Java, and the experiments are conducted on Windows Server 2008 with 16-core CPU, 256 GB main memory.

4.1.1 Data sets

We use three real-world networks to evaluate our algorithm, which are Citeseer, Cora and Flickr. Citeseer includes 3312 nodes and 4372 edges; Cora includes 2708 nodes

Table 3 Data sets

Data set	$\|V\|$	$\|E\|$
Citeseer	3312	4732
Cora	2708	5429
Flickr	4546	383,697

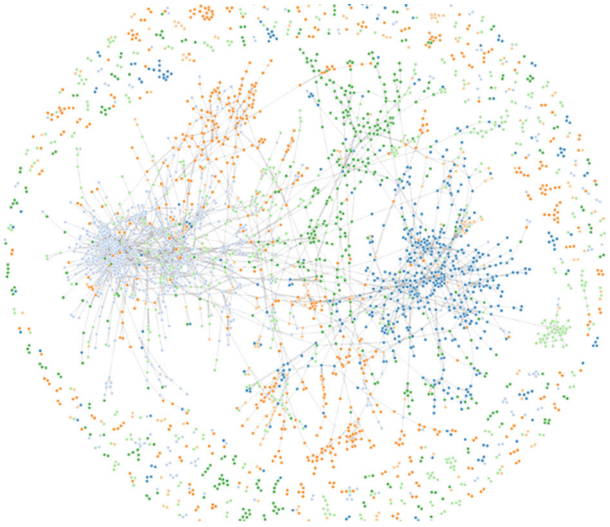


Fig. 12 Citeseer with six communities

and 5429 edges; and Flickr includes 4546 nodes and 383,697 edges. The information of data sets is shown in Table 3.

1. Citeseer

In Citeseer, all nodes are divided into six classes, namely agents, AI (artificial intelligence), DB (database), IR (information retrieval), ML (machine learning) and HCI (human–computer interaction), which represent the ground-truth communities in Citeseer. The visualization of the Citeseer network is shown in Fig. 12 where different colors represent different classes or communities. The nodes in Citeseer are sorted by descending order of the degree values, and the 1st-20th nodes and their degrees are shown in Table 4.

2. Cora

In Cora, all nodes are divided into seven classes, namely case based, genetic algorithms, neural networks, probabilistic methods, reinforcement learning, rule learning and theory, which represent the ground-truth communities in Cora. The nodes and edges in Cora are shown in Fig. 13 where different colors represent different classes or communities.

The nodes in Cora are sorted by descending order of their degrees, and the 1st−18th nodes and their degrees are shown in Table 5.

Table 4 The 1st−18th nodes and their degrees in Citeseer

Node ID	Degree
brin98anatomy	100
rao95bdi	51
chakrabarti98automatic	35
bharat98improved	34
lawrence98searching	30
craven98learning	29
87928	28
lawrence00context	27
soderland99learning	23
jennings95controlling	23
kitano97robocup	22
aha91casebased	21
howe97savvysearch	21
lawrence98context	20
jantke93casebased	19
dean99finding	19
marquez00machine	19
decker95environment	18

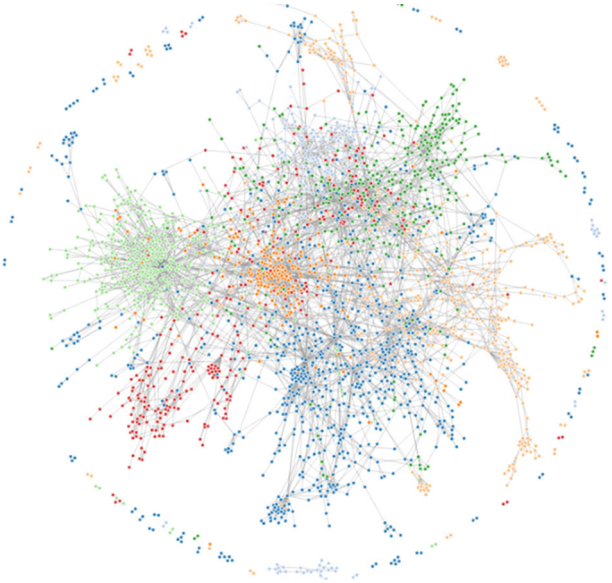


Fig. 13 Cora with seven communities

Table 5 The 1st−18th nodes and their degrees in Cora

Node ID	Degree
35	169
6213	79
1365	74
3229	65
910	46
114	42
4330	40
3231	37
1272	34
19621	34
4584	32
6214	32
2440	31
2665	31
887	30
24966	29
8224	26
82920	23

3. Flickr

For the data set Flickr, nodes are sorted by descending order of their degrees, and 18 nodes from different degree ranks are shown in Table 6. As there are no information about the ground-truth communities for Flickr, we only use it to evaluate the time cost for our method AGAR.

4.1.2 Evaluation metrics

In this paper, the values of F-score of predicted communities and ground-truth communities in different data sets are computed to evaluate the quality of the predicted communities [16].

Given a community p from a set of predicted communities P and a real community g from a set of ground-truth communities G, several concepts are defined as follows.

Definition 4 (*Precision*) In the field of information retrieval, precision is the fraction of retrieved documents that are relevant to the query. The precision of predicted community p and ground-truth community g is:

$$Precision = \frac{|p \cap g|}{|p|} \tag{3}$$

Definition 5 (*Recall*) Recall in information retrieval is the fraction of the documents that are relevant to the query that are successfully retrieved. The recall of predicted

Table 6 The 1st-18th nodes and their degrees in Flickr

Node ID	Degree
2288245961	1087
2441604368	723
67558976	629
2612205228	567
2180780268	518
2587388200	476
618163447	436
2487262614	396
2337003408	360
2311624965	324
860346638	288
2592913664	252
2293714244	216
2087179582	180
2363794360	144
2534860444	108
2274327896	72
544162745	36

community p and ground-truth community g is:

$$Recall = \frac{|p \cap g|}{|g|} \tag{4}$$

Definition 6 (f-$score$) f-$score$ that combines precision and recall is the harmonic mean of precision and recall. The f-$score$ of predicted community p and ground-truth community g is:

$$f\text{-}score = \frac{2Precision \cdot Recall}{Precision + Recall} \tag{5}$$

Definition 7 ($F(p, G)$) $F(p, G)$ is the max value of predicted community p and each ground-truth community g from G. The $F(p, G)$ of predicted community p in ground-truth communities G is:

$$F(p, G) = max_{g \in G} f\text{-}score(p, g) \tag{6}$$

Definition 8 (F-$score$) F-$score$ represents the weighted average F-$score$ value of predicted communities. The F-$score$ of predicted communities P in ground-truth communities G is:

$$F\text{-}score = \sum_{p \in P} \frac{|p|}{|V|} F(p, G) \tag{7}$$

Thus, the $F\text{-}score$ value for the predicted communities $\mathcal{P}$ w.r.t. the ground-truth communities $\mathcal{G}$ is:

$$F\text{-}score = \sum_{p \in \mathcal{P}} \left(\frac{|p|}{|V|} max_{g \in \mathcal{G}} \left(\frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \right) \right) \tag{8}$$

4.1.3 Baselines

We use the following two methods as the baselines which also aim to find out the communities for the given nodes.

- k-**truss** [16] constructs the TCP-index to find the communities for the given nodes based on the k-truss.
- **ACQ** [11] is a community search method to find out the communities for the given nodes by constructing the CL-tree index based on k-core and the node attributes.

4.2 Experimental results

4.2.1 Quality evaluation

1. The visualization for TA-graph

In this part, we compare the TA-graph shown in Figs. 14 and 15 with the initial graph shown in Figs. 12 and 13 to demonstrate the necessity for the graph refining operation. In the TA-graph, nodes in the same community are more densely connected and nodes between different communities are more sparsely connected compared with the nodes in initial graph G.

2. The comparison with ground-truth communities

In this part, we use the metric F-score on two real-world networks (i.e., Citeseer and Cora) which are both sparse networks, to evaluate the quality of communities using the k-truss [16], ACQ [11] and our method AGAR, respectively, as shown in Fig. 16. In Citeseer, we select the 1st−8th nodes in Table 4 as the query nodes and then compute the average value of F-score. When $k = 3-16$, the average values of F-score using k-truss and AGAR are shown in Fig. 16a. In Cora, we select the 1st−7th nodes in Table 5 as the query nodes and then compute the average value of F-score. When $k = 3-16$, the average values of F-score using k-truss and AGAR are shown in Fig. 16b.

The F-score value for AGAR is much higher than that of k-truss and ACQ, which indicates that the communities obtained by our AGAR are more accurate than those by the k-truss and ACQ. AGAR can find more accurate communities than baselines, because we construct the AttrTCP-index based on the structure of TA-graph which can reveal more obvious community structure. Compared with k-truss, AGAR makes full use of the information of node attributes to construct the accurate index. The ACQ aims to search the communities which satisfy the requirements in the aspect of topology and that of the node attributes at the same time. The nodes that are similar to the query

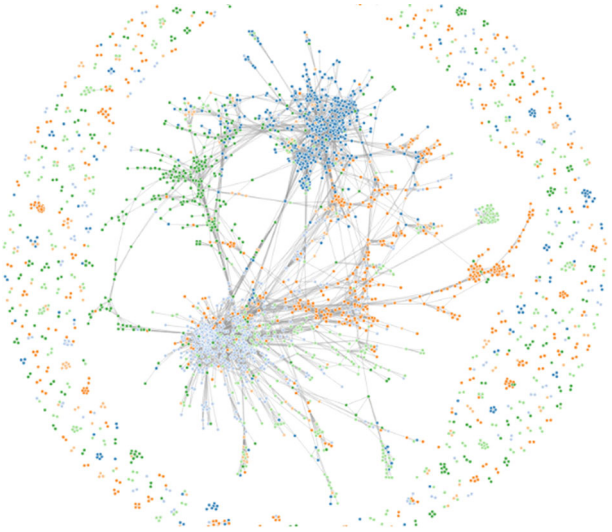


Fig. 14 TA-graph for Citeseer

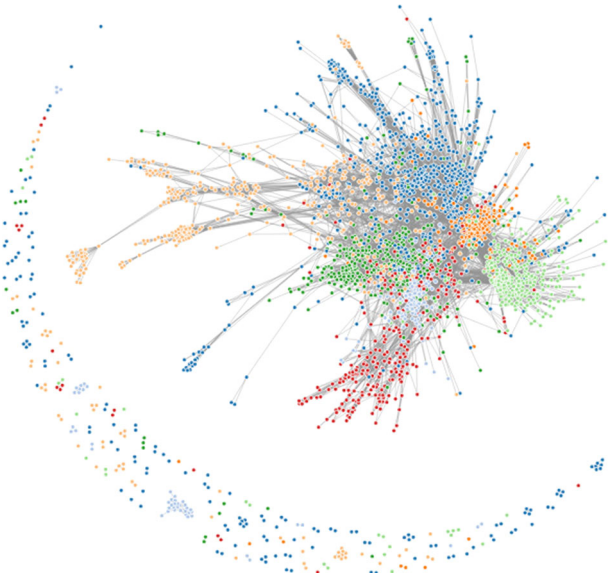


Fig. 15 TA-graph for Cora

node from one of the above two respects are filtered in the querying process of ACQ. Different from the ACQ, AGAR finds the communities which meet the requirements in topology or in node attributes, which can find more complete communities for the given node, especially in sparse networks. With the increase in k, the F-score value using AGAR first increases, then decreases on Citeseer and decreases on Cora.

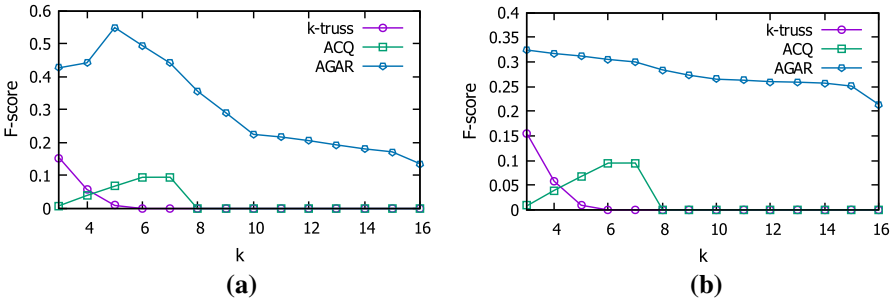


Fig. 16 The F-score values with k in two data sets. **a** Citeseer, **b** Cora

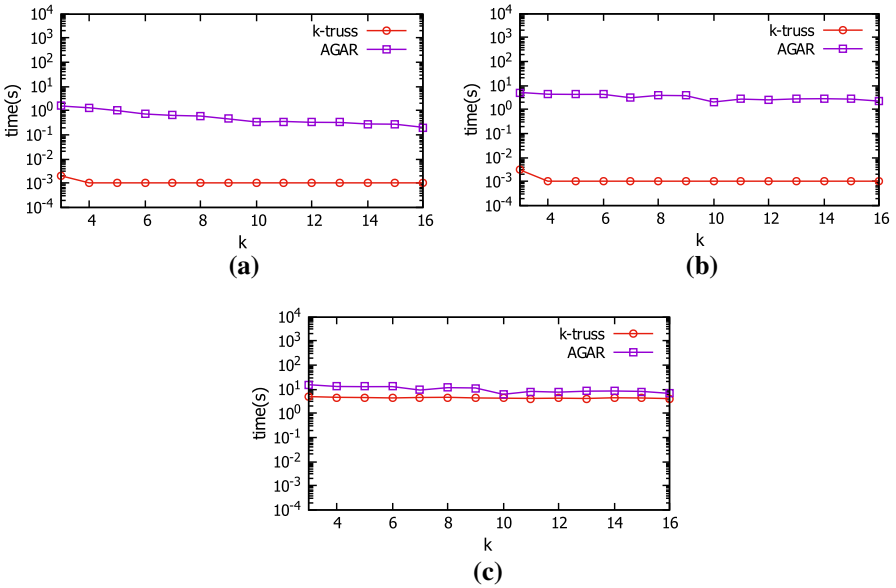


Fig. 17 Query time (in s) of two methods for different k. **a** Citeseer, **b** Cora, **c** Flickr

Because both Citeseer and Cora are sparse data sets, the F-score value of the k-truss method decreases rapidly when k is larger than 5.

4.2.2 Efficiency analysis

In this subsection, we evaluate time efficiency for k-truss and AGAR in the process of querying, index construction and graph refining.

To evaluate the time cost of query processing, we firstly vary the parameter k to test query time for k-truss and AGAR in three data sets. For each network, we vary $k = 3-16$ and test the average querying time of the query nodes in Tables 4, 5 and 6. Figure 17 shows the querying time of each method when we vary the parameter k. With the increase in k, query time decreases because the bigger the value of k is, the smaller the index size is.

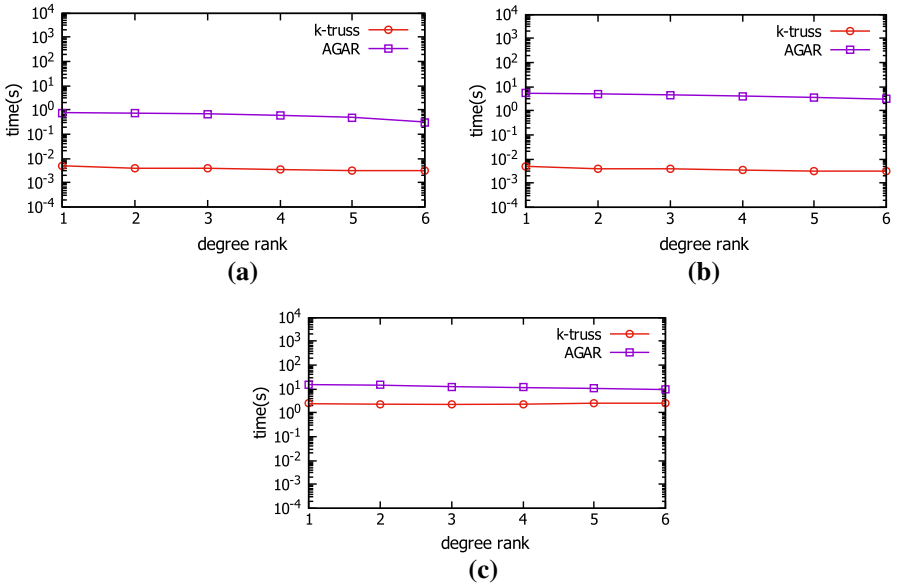


Fig. 18 Query time (in s) of different methods for different degree rank values. **a** Citeseer, **b** Cora, **c** Flickr

Table 7 Time costs of constructing index structures of the two methods

Data set	K-Truss (s)	AGAR(s)
Citeseer	0.93	6.67
Cora	1.23	36.62
Flickr	943.13	4265

Table 8 Time costs of graph refining on the three data sets

Data set	AGAR(s)
Citeseer	34.81
Cora	15.83
Flickr	83,248

Then, we select query nodes with different degree ranks to test the querying time. In Tables 4, 5 and 6, we sort the nodes in descending order of their degrees. For each network, we partition the nodes into six equal-sized buckets and test the average time cost for each bucket of query nodes which are shown in Fig. 18.

The k-truss is faster than AGAR because after the graph refining, the TA-graph is denser than the initial graph G. Thus, the size of AttrTCP-index is larger than that of the index directly constructed by k-truss. This guarantees that the communities found by AGAR are much larger, more complete and more accurate than the communities searched by the k-truss. In addition, the querying time for AGAR and k-truss is in the same time magnitude. Thus, AGAR is as efficient as k-truss.

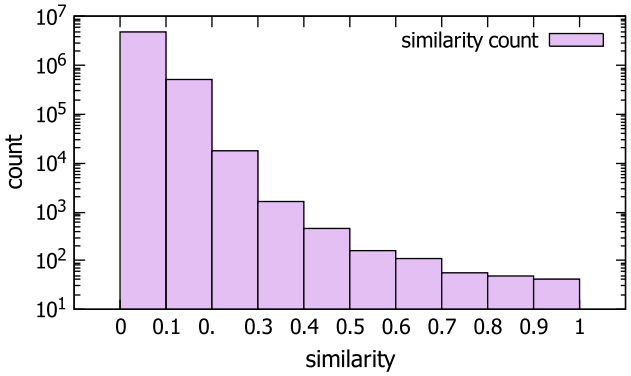


Fig. 19 Distribution of attribute-based similarity of Citeseer

The time cost of constructing index structures of the two comparative methods for three data sets is shown in Table 7. The time cost of graph refining of AGAR for the data sets is shown in Table 8. Considering both graph refining and index building are offline, the performance of these methods is also acceptable.

4.2.3 Parameter analysis

In this part, we vary the values of $simR$, α and β to demonstrate the effect of these parameters on the performance of our method and indicate the rationality for setting $r = l = m$, $\alpha = 1/2$, $\beta = 1/3$. In Citeseer and Cora, we select the 1st−8th nodes in Table 4 and the 1st−7th nodes in Table 5 as the query nodes, respectively. We compute the average values of F-score for the query nodes to analyze the effects of $simR$, α and β on the accuracy of predicted communities.

First of all, we plot the distribution of attribute-based similarity of Citeseer, Cora and Flickr as shown in Figs. 19, 20 and 21, respectively. For Citeseer and Cora, we choose $simL = 0$ because these two data sets are sparse networks, which is obviously suitable. In the following, we test the effect of $simR$ on the values of F-score on Citeseer and Cora. Figure 22 shows the values of F-score for different values of $simR$ on these two data sets using AGAR when $k = 5$, $simL = 0$, $\alpha = 1/2$, and $\beta = 1/3$. With the increase in the values of $simR$, the values of F-score are decreasing. When $simR$ is set to the value higher than 0.3, the F-score is stable because the most similarities for Citeseer and Cora are in the range from 0 to 0.3 as shown in Figs. 19 and 20. When $simR = 0.1$, we get the max value of F-score. Thus, $simR = 0.1$ seems to be the most suitable value when $r = l = m = 4732$ for Citeseer and when $r = l = m = 5429$ for Cora.

The values of F-score for different values of α using AGAR on Citeseer and Cora are shown in Fig. 23 where $k = 5$, $simR = 0.1$, $simL = 0$, and $\beta = 1/3$. For these two data sets, with the increase in α, the values of F-score first increase, then decrease and reach maximum when α is 1/2. As a result, setting $\alpha = 1/2$ is suitable. When $k = 5$, $simR = 0.1$, $simL = 0$, and $\alpha = 1/2$, the average values of F-score for different values of β using AGAR for the query nodes are shown in Fig. 24. For

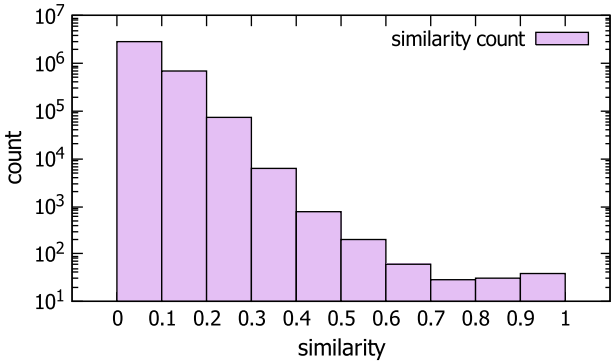


Fig. 20 Distribution of attribute-based similarity of Cora

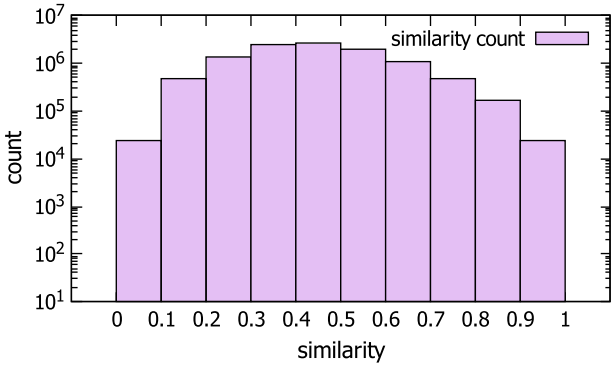


Fig. 21 Distribution of attribute-based similarity of Flickr

Citeseer, the values of F-score increase first and then decrease with the increase in β. The value of F-score reaches the maximum when β is 1/3. For Cora, the values of F-score decrease with the increase in β. There are little difference between the values of F-score when β equals 1/5, 1/4 and 1/3. The values of F-score decrease when the values of β are higher than 1/3. Thus, in our method, choosing β to be 1/3 is reasonable.

In addition, the values of F-score change slowly with different values of $simR$, α and β, which indicates the stability of our method.

4.2.4 Case study

In Citeseer, the given query node $agarwal01time$ whose degree is six belongs to "database" community. The predicted communities using k-truss and AGAR are shown in Fig. 25. The community using AGAR is more accurate and complete than k-truss. AGAR finds out the five nodes which k-truss also finds out as shown in Fig. 25a. Besides, our AGAR finds out 32 nodes as shown in Fig. 25b which k-truss does not find out. The community comparison of two methods indicates that AGAR can find out more accurate and complete communities than k-truss.

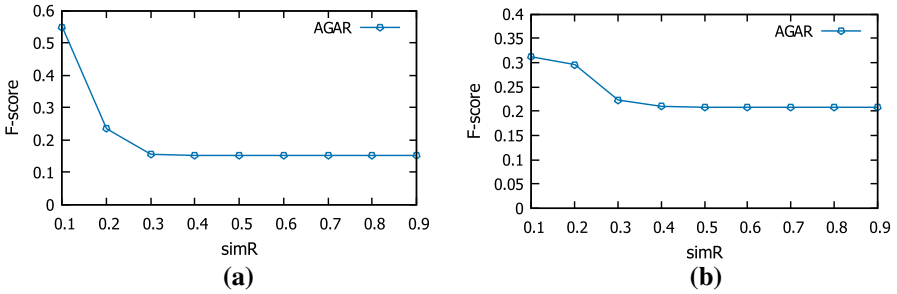


Fig. 22 F-score of AGAR for different simR values. **a** Citeseer, **b** Cora

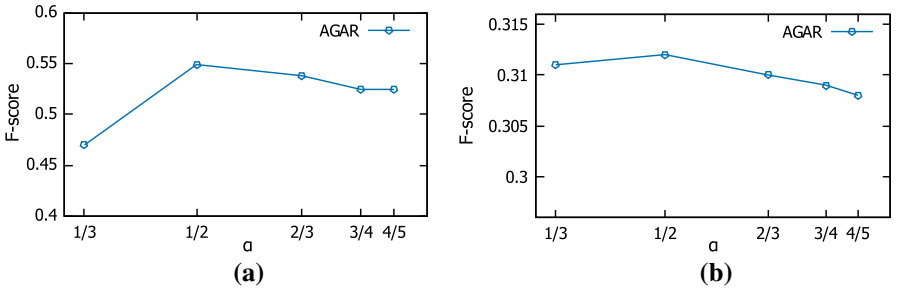


Fig. 23 F-score of AGAR for different α values. **a** Citeseer, **b** Cora

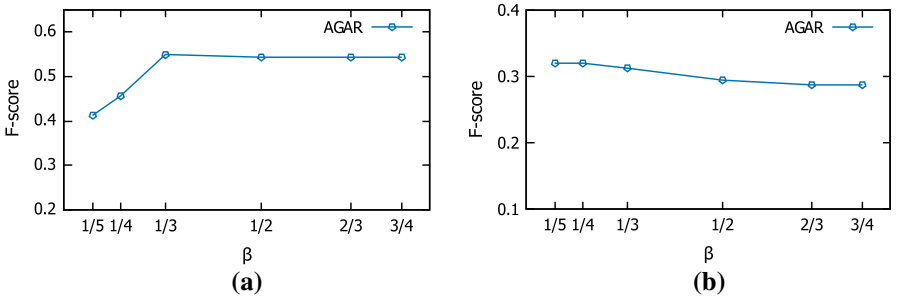


Fig. 24 F-score of AGAR for different β values. **a** Citeseer, **b** Cora

5 Related work

5.1 Community detection with node attributes

Community detection aims to find out the densely connected subgraphs according to the relations in a network. This issue is adequately and extensively studied in the literature [3,12–14,23,25,28,36,39,40]. The popular community detection methods include hierarchy clustering [6,22,23,26], label propagation [19,27,38], matrix blocking [4,34] and graph skeleton clustering [15].

Besides, various community detection methods with node attributes have also been proposed and studied. One of them is the probabilistic modeling method [20,33,41,42].

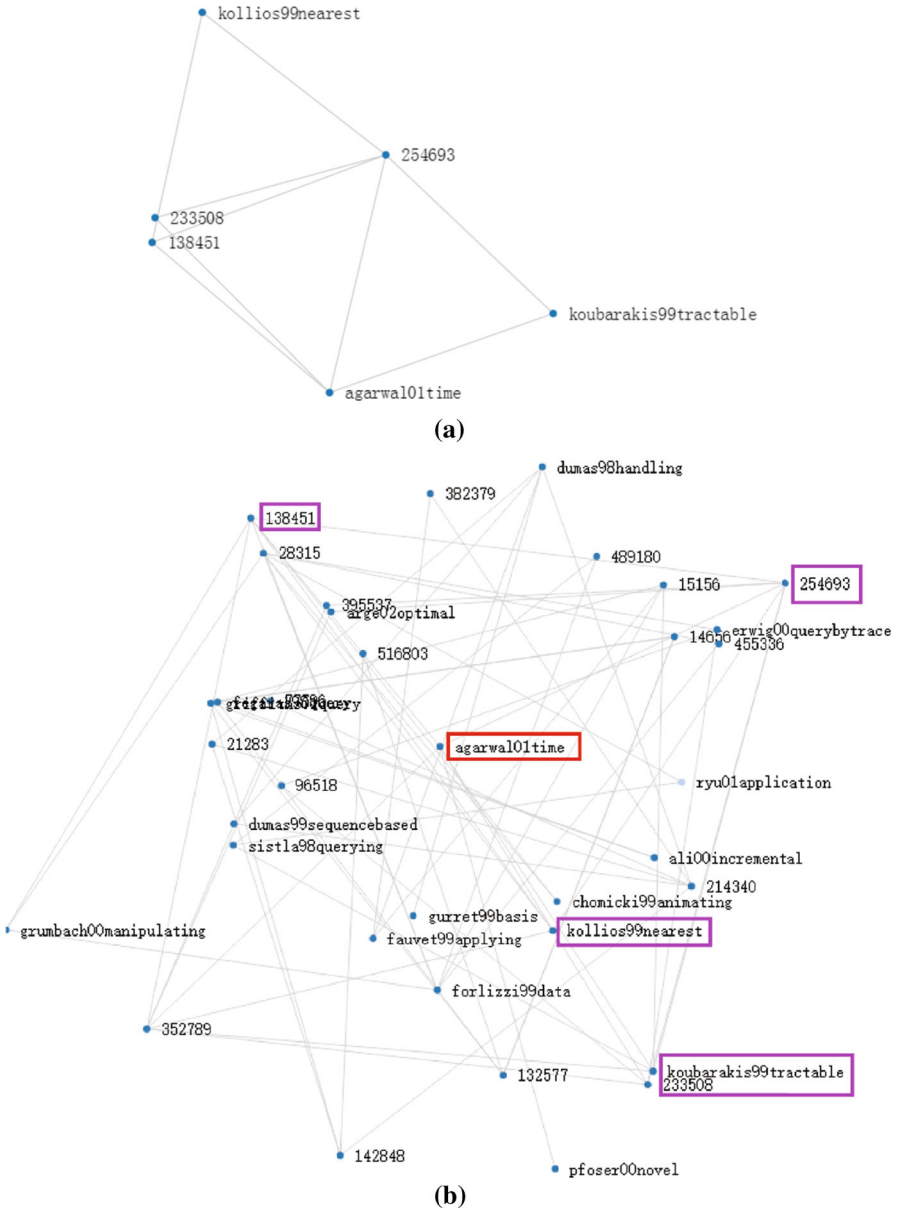


Fig. 25 Comparison of communities using two methods. **a** Community result using k-truss method, **b** community result using AGAR method

Liu et al. [20] developed a Bayesian hierarchical approach to perform the topic modeling and detect the communities in a unified framework. Xu et al. [41] developed a Bayesian probabilistic model for the attributed network. The model provided a principled and natural framework to capture both structure and attribute aspects of a network,

which avoided the artificial design of a distance measure. Sun et al. [33] designed a probabilistic model to cluster the objects of different types into a common hidden space, according to a user-specified set of attributes, as well as the links from different relations. Yang et al. [42] proposed the CESNA, a probabilistic generative model that combines community memberships, the network topology and node attributes to find the communities. Another typical method is the heuristic method [1,10,21,29,44] which combines the network topology and node attribute information. Zhou et al. [44] proposed a graph clustering algorithm, SA-cluster, based on both structure and attribute similarities through a unified distance measure. Akoglu et al. [1] proposed a parameter-free method, PICS, mining the attributed network. Ruan et al. [29] presented CODICIL, a graph simplification algorithm to leverage both network topology and node attribute to identify and retain important edges in a network.

5.2 Community search

Community search, also called local community detection, is a problem of finding the groups of densely connected nodes containing the query node or a group of nodes. Different from the problem of community detection which has been adequately and extensively studied, community search, as a novel emerging problem, has attracted much attention in recent years. In 2005, Clauset et al. [5] proposed the problem of local community detection. The definition of community search was firstly proposed by Sozio et al. [32] in 2010.

The existing community search methods can be divided into four main categories, which are k-clique-based method [9], k-core-based method [2,5,32], k-truss-based method [16,17] and densest subgraph-based method [7,37]. The k-clique-based method utilizes the complete subgraphs in the network to find communities. As a typical k-clique-based method, Cui et al. [9] proposed an α-adjacency γ-quasi-k-clique model to solve the overlapping community search problem. The k-core-based method is the most widely used community search methods and focuses on the degrees of nodes. In [32], Sozio et al. proposed a linear-time algorithm aiming to find the communities that contain the query nodes based on the k-core model. In [5], Clauset defined a fast agglomerative algorithm which can maximize the local modularity in a greedy fashion. Barbieri et al. [2] constructed the ShellStruct to query the communities for the given nodes based on the k-core model. The k-truss-based community search method is newly proposed and focuses on the triangles for the nodes. Huang et al. [16] presented a k-truss community model and then proposed the TCP-index based on the k-truss model to effectively query the communities containing given node. Then, Huang et al. [17] presented a greedy algorithmic framework to achieve 2-approximation to the optimal solution for the community search problem. The last kind of community search method is the densest subgraph-based method. Wu et al. [37] developed a node weighting scheme to reduce the free rider effect and provided algorithms to solve the query biased densest connected subgraph (QDC) problem. Conde et al. [7] proposed a local community detection algorithm based on local optimizations to approximate the maximal α-consensus local community of a given node. However, existing community

search methods only use the network topology information to find communities and overlook the rich information of node attributes.

The most similar work to ours is [11]. In [11], the authors explored both network topology and node attributes to find out the communities for the query nodes. However, there are major differences between ours and [11] as follows. The methods proposed in [11] aim to search the communities which meet the requirements of k-core in topology and common keywords representing the node attributes at the same time. The nodes that are similar to the query node from one of the two respects are filtered in the querying process. However, this is not suitable for the sparse networks where the ratio of the numbers of edges and nodes is lower than 5:1. In this paper, our method can find more accurate communities for the given nodes, especially in the sparse networks, by strengthening the relations between nodes from the respect of topology-based similarity and that of attribute-based similarity. At first, we propose a graph refining technique to construct the TA-graph which reflects both the topology-based similarity and attribute-based similarity. We further explore the effect of node attributes in the construction of AttrTCP-index to improve the structure of TA-graph. By querying the AttrTCP-index, we can select the scale of communities according to our personal preference. Besides, the communities by our method AGAR are more accurate especially for the sparse networks.

6 Conclusions

Community search aims to find out the communities containing the given nodes. In this paper, we propose an attribute-based community search method with graph refining called AGAR. We firstly construct the TA-graph by computing the topology-based similarity and the attribute-based similarity. Then, we compute the AttrSup for each edge and construct the AttrTCP-index. Finally, by querying the AttrTCP-index, the communities for the given node can be found efficiently. Experiments on real-world data sets indicate that our method AGAR can find more accurate communities.

Acknowledgements This work is supported in part by the National Natural Science Foundation of China (No. 61373023) and the China National Arts Fund (No. 20164129).

References

1. Akoglu L, Tong H, Meeder B, Faloutsos C (2012) PICS: parameter-free identification of cohesive subgroups in large attributed graphs. In: SDM. Citeseer, pp 439–450
2. Barbieri N, Bonchi F, Galimberti E, Gullo F (2015) Efficient and effective community search. Data Min Knowl Discov 29(5):1406–1433
3. Cai Z, He Z, Guan X, Li Y (2016) Collective data-sanitization for preventing sensitive information inference attacks in social networks. IEEE Trans Dependable Secure Comput. doi:10.1109/TDSC.2016.2613521
4. Chen J, Saad Y (2012) Dense subgraph extraction with application to community detection. IEEE Trans Knowl Data Eng 24(7):1216–1230
5. Clauset A (2005) Finding local community structure in networks. Phys Rev E 72(2):026,132
6. Clauset A, Newman ME, Moore C (2004) Finding community structure in very large networks. Phys Rev E 70(6):066,111

7. Conde CP, Ngonmang B, Viennet E et al (2015) Approximation of the maximal alpha—consensus local community detection problem in complex networks. In: 2015 11th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS). IEEE, pp 314–321
8. Cui W, Xiao Y, Wang H, Lu Y, Wang W (2013) Online search of overlapping communities. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. ACM, pp 277–288
9. Cui W, Xiao Y, Wang H, Wang W (2014) Local search of communities in large graphs. In: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data. ACM, pp 991–1002
10. Ester M, Ge R, Gao BJ, Hu Z, Ben-Moshe B (2006) Joint cluster analysis of attribute data and relationship data: the connected k-center problem. In: SDM, vol 6. SIAM, pp 25–46
11. Fang Y, Cheng R, Luo S, Hu J (2016) Effective community search for large attributed graphs. Proc VLDB Endow 9(12):1233–1244
12. Fortunato S (2010) Community detection in graphs. Phys Rep 486(3):75–174
13. Fortunato S, Hric D (2016) Community detection in networks: a user guide. Phys Rep 659:1–44
14. Han M, Yan M, Cai Z, Li Y, Cai X, Yu J (2016) Influence maximization by probing partial communities in dynamic online social networks. Trans Emerg Telecommun Technol. doi:10.1002/ett.3054
15. Huang J, Sun H, Song Q, Deng H, Han J (2013) Revealing density-based clustering structure from the core-connected tree of a network. IEEE Trans Knowl Data Eng 25(8):1876–1889
16. Huang X, Cheng H, Qin L, Tian W, Yu JX (2014) Querying *k*-truss community in large and dynamic graphs. In: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data. ACM, pp 1311–1322
17. Huang X, Lakshmanan LV, Yu JX, Cheng H (2015) Approximate closest community search in networks. Proc VLDB Endow 9(4):276–287
18. Kim S, Lee W, Arora NR, Jo TC, Kang SH (2012) Retrieving keyworded subgraphs with graph ranking score. Expert Syst Appl 39(5):4647–4656
19. Leung IX, Hui P, Lio P, Crowcroft J (2009) Towards real-time community detection in large networks. Phys Rev E 79(6):066,107
20. Liu Y, Niculescu-Mizil A, Gryc W (2009) Topic-link LDA: joint models of topic and author community. In: Proceedings of the 26th Annual International Conference on Machine Learning. ACM, pp 665–672
21. Moser F, Colak R, Rafiey A, Ester M (2009) Mining cohesive patterns from graphs with feature vectors. In: SDM, vol 9. SIAM, pp 593–604
22. Newman ME (2004) Fast algorithm for detecting community structure in networks. Phys Rev E 69(6):066,133
23. Newman ME, Girvan M (2004) Finding and evaluating community structure in networks. Phys Rev E 69(2):026,113
24. Nguyen HV, Bai L (2011) Cosine similarity metric learning for face verification. In: Computer Vision—ACCV 2010. Springer, pp 709–720
25. Palla G, Derényi I, Farkas I, Vicsek T (2005) Uncovering the overlapping community structure of complex networks in nature and society. Nature 435(7043):814–818
26. Radicchi F, Castellano C, Cecconi F, Loreto V, Parisi D (2004) Defining and identifying communities in networks. Proc Natl Acad Sci USA 101(9):2658–2663
27. Raghavan UN, Albert R, Kumara S (2007) Near linear time algorithm to detect community structures in large-scale networks. Phys Rev E 76(3):036,106
28. Rosvall M, Bergstrom CT (2008) Maps of random walks on complex networks reveal community structure. Proc Natl Acad Sci USA 105(4):1118–1123
29. Ruan Y, Fuhry D, Parthasarathy S (2013) Efficient community detection in large networks using content and links. In: Proceedings of the 22nd International Conference on World Wide Web. International World Wide Web Conferences Steering Committee, pp 1089–1098
30. Schank T (2007) Algorithmic aspects of triangle-based network analysis. PhD in computer science, University Karlsruhe 3
31. Shang J, Wang C, Wang C, Guo G, Qian J (2016) AGAR: an attribute-based graph refining method for community search. In: Proceedings of the 6th International Conference on Emerging Databases, pp 80–81
32. Sozio M, Gionis A (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. ACM, pp 939–948 (2010)

33. Sun Y, Aggarwal CC, Han J (2012) Relation strength-aware clustering of heterogeneous information networks with incomplete attributes. Proc VLDB Endow 5(5):394–405
34. Tsourakakis C, Bonchi F, Gionis A, Gullo F, Tsiarli M (2013) Denser than the densest subgraph: extracting optimal quasi-cliques with quality guarantees. In: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, pp 104–112 (2013)
35. Wang J, Cheng J (2012) Truss decomposition in massive networks. Proc VLDB Endow 5(9):812–823
36. Wang M, Wang C, Yu JX, Zhang J (2015) Community detection in social networks: an in-depth benchmarking study with a procedure-oriented framework. Proc VLDB Endow 8(10):998–1009
37. Wu Y, Jin R, Li J, Zhang X (2015) Robust local community detection: on free rider effect and its elimination. Proc VLDB Endow 8(7):798–809
38. Xie J, Szymanski BK, Liu X (2011) Slpa: uncovering overlapping communities in social networks via a speaker–listener interaction dynamic process. In: 2011 IEEE 11th International Conference on Data Mining Workshops. IEEE, pp 344–349
39. Xie J, Kelley S, Szymanski BK (2013) Overlapping community detection in networks: the state-of-the-art and comparative study. ACM Comput Surv (CSUR) 45(4):43
40. Xin X, Wang C, Ying X, Wang B (2017) Deep community detection in topologically incomplete networks. Phys A 469:342–352
41. Xu Z, Ke Y, Wang Y, Cheng H, Cheng J (2012) A model-based approach to attributed graph clustering. In: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. ACM, pp 505–516 (2012)
42. Yang J, McAuley J, Leskovec J (2013) Community detection in networks with node attributes. In: 2013 IEEE 13th International Conference on Data Mining (ICDM). IEEE, pp 1151–1156 (2013)
43. Zhang M, Hu H, He Z, Gao L, Sun L (2015) Efficient link-based similarity search in web networks. Expert Syst Appl 42(22):8868–8880
44. Zhou Y, Cheng H, Yu JX (2009) Graph clustering based on structural/attribute similarities. Proc VLDB Endow 2(1):718–729