2018 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computing, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovations

Attributed Community Search in Dynamic Networks

Jiahao Zhang
School of Computer Science and Technology/Suzhou Institute for Advanced Study, University of Science and Technology of China, P.R.china
jhcheung@mail.ustc.edu.cn

Kai Han
School of Computer Science and Technology/Suzhou Institute for Advanced Study, University of Science and Technology of China, P.R.china
hankai@ustc.edu.cn

Junxiao Han
College of Computer Science and Technology, Zhejiang University, Hangzhou, China
junxiaohan@zju.edu.cn

Feiyang Li
School of Computer Science and Technology/Suzhou Institute for Advanced Study, University of Science and Technology of China, P.R.china
mmmwhy@mail.ustc.edu.cn

Shasha Li
School of Computer Science and Technology/Suzhou Institute for Advanced Study, University of Science and Technology of China, P.R.china
lisa1990@mail.ustc.edu.cn

***Abstract*—Recently, community search has been studied a lot. Given a query vertex in a graph, the problem is to find a meaningful community that contains the vertex. It has drawn extensive attention from both academia and industry. However, most of the existing solutions don't take the features of nodes into consideration. And other algorithms, which can be used in attributed graphs, often need the help of index structures. In real life, social networks are usually rapidly evolving entities in which the features of nodes and the structure of the graph may change rapidly over time, and these methods need to constantly update or even rebuild the index. In other words, it is costly and time-consuming. In this paper, we propose an *attributed community search* (ACS) strategy, which returns a community that satisfies both structure cohesiveness (i.e., its vertices are tightly connected) and keyword cohesiveness (i.e., its vertices have similar keywords with the query node), without the support of index structures (e.g., tree structure). ACS achieves efficient and accurate community search and can be applied in dynamic graphs. The effectiveness and efficiency of our attributed community search strategy is verified by extensive experiments on several real networks with millions of nodes.**

***Index Terms*—Community search, Social networks, Attributed graph, Graph mining**

1. Introduction

With the development of large social networks, the topic of attributed graphs has received considerable attention from industry and research communities [1]–[16]. In these graphs, a node (or user) is often associated with keywords or attributes. Figure 1 depicts an attributed graph, and each user has a keyword set which describes the interests of music.

Kai Han is the corresponding author.

One of the most important tasks when studying networks is to identify the communities. In this paper, we discover community from both edge structure and node attributes, an accurate and efficient algorithm for searching community in attributed graphs. Given an attributed graph G and a vertex $v \in G$, the Attributed Community Search (ACS) algorithm can find a subgraph of G, called attribute-aware community (or AAC). The AAC satisfies both structure cohesiveness and keyword cohesiveness. The common attributes of vertices can reveal the theme of the community. Figure 1 illustrates an AAC (marked in red), which is a connected subgraph with vertex degree 3; people {John, Mike, Bob, Jack} in the community all like rock and blues music. If we do not consider the attributes attached to the vertices, then vertices {John, Mike, Bob, Jack, Alex, Tom} are intuitively more likely to be in one community. But Alex and Tom have quite different music hobbies with others. Thus the use of keyword information can significantly improve the effectiveness and accuracy of the communities retrieved.

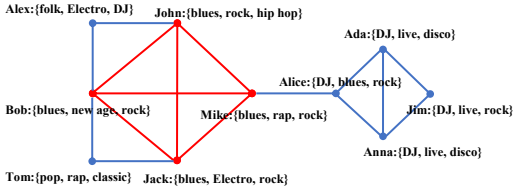


Figure 1: Example of an attributed graph.

Communities are ubiquitous in attributed graphs such as knowledge bases and social networks, which can be used in a lot of applications. Typical applications include:

- *Control of infectious disease.* If we know that a person has been infected with an infectious disease, and then who will most likely be affected? Obviously,

978-1-5386-9380-3/18/$31.00 ©2018 IEEE
DOI 10.1109/SmartWorld.2018.00162

those who are in the same community as the infected person. We can attain the aim to control the diffusion of epidemic by means of isolating these individuals.
- *Advertising on social networks.* If we know that a person is interested in a particular type of advertisement. People in the same community with him often have common interests. So we may push the same type of advertisements to the members in the community.
- *Friends recommendation on social networks.* Friends recommendation system is common in social networks, such as Facebook. If we want to recommend candidate friends to a specific user, we can recommend those who are in the same community but are not yet friends.

We now present more details about our ACS method. The main contributions of our solutions can be summarized as follows.

(1) **Adaptability for the change of networks.** In social networks (e.g., Facebook, Brightkite and Foursquare), a user's features (e.g., interests, location) often changes. Moreover, the link relationship between vertices also evolves over time. So the existing Community Search algorithms based on offline computation (such as graph clustering or index structures), may easily lose the effectiveness and freshness after a short while. On the contrary, our ACS algorithm can adapt to such dynamic social networks easily.

(2) **Personalization.** The user of our ACS can control the semantics of the AAC, by specifying the query vertex's keywords set S. Intuitively, S decides the meaning of the community based on the user's need.

Challenges. The ACS problem is very challenging. We mainly face the following two questions: Firstly, to effectively implement ACS, we must determine what a good AAC is. For this question, we adopt *minimum degree* to measure the closeness of the vertices in the community, which is widely used as structure cohesiveness metrics [17]–[19]. This measure requires that the degree of each vertex in the community to be at least k. And we use the number of shared keywords with query node to measure the keyword cohesiveness. The shared keywords represent the common features among vertices, which can reveal the theme of the community.

The second question is how to make ACS algorithm efficient? The cohesiveness criteria of the community can be difficult to handle. A straightforward method is first to enumerate all the subsets of the query vertex's keyword set, and then find a subgraph, which satisfies the minimum degree constraint and has the most number of common keywords. But this solution is unacceptable, because it has a complexity exponential to the size l of q's keyword set.

To solve this problem, most of the existing attributed community search algorithms such as [9] [10] [20] use index structure to speed up the search process. But when the graph is large, the indexing process may take at least one hour or even longer. Therefore, these solutions are inefficient in dynamic graphs because we need to constantly update the index. In this paper, we introduce a novel community model called attribute-aware community, which can satisfy both structure cohesiveness and keyword cohesiveness. Based on the new community model, we use local search strategy to find community in attributed graphs, which searches in the neighborhood of the query node. Each query is based on the network structure of the current moment, and the result can be obtained within tens of milliseconds. When the graph structure changes, we can quickly get new and effective results by re-querying.

Organization. We discuss the related work in Section 2 and define the ACS problem formally in Section 3. Section 4 presents the proposed query algorithms. We report the experimental results in Section 5. Section 6 concludes this work.

2. RELATED WORK

The community retrieval problems can generally be classified into *community detection* (CD) and *community search* (CS).

Community detection (CD). Community detection aims to find all the communities from an entire graph, and it has been studied a lot in recent years. Classical solutions, such as [17] [21], only employ link-based analysis into consideration to obtain communities. However, they do not consider the background information associated with vertices in the graph. Recently, there have been some works that focus on the study of attributed graphs, and they use clustering methods to identify all the communities [22]–[25]. There are some differences between CD algorithms and our ACS method. First of all, CD algorithms are generally costly and slow, as they usually detect all the communities from an entire network. Second, it is not clear how they can be adapted for ACS.

Community search (CS). Recently, there is another type of research on communities, called community search (CS). The goal of CS is to obtain communities based on a query request. For instance, given a vertex q, several effective methods [18] [19] [26] [27] [28] have been proposed to obtain a community that contains q. Other well known community definitions, including k-clique [26], k-truss [28] and connectivity [29], have also been used to search communities in an online manner. But these works assume non-attributed graphs. There also exists works [20] [9] [10] to search communities from attributed graphs. However, these algorithms require the assistance of index structures. In other words, an offline pre-computation is needed every time the structure of the graph evolves, which is costly and time-consuming. In addition, there is another algorithm FocusCO [11] that can be used in attributed graphs and does not require an index. But it needs to detect all the vertices in the graph, and the query speed is slow. Thus, it is significant to design an efficient algorithm to find communities from dynamic attributed graphs.

3. PROBLEM DEFINITION

Let $G(V, E)$ denote an undirected and attributed graph with edge set E and vertex set V. Each vertex $v \in V$ is associated with a set of keywords, $Key(v)$. Moreover, we use $deg_G(v)$ to denote the degree of vertex v in G. Table 1 lists the notations used in this paper.

TABLE 1: **Notations and meanings**

Notation	Meaning
$G(V, E)$	An attributed graph with vertex set V and edge set E
$deg_G(v)$	The degree of vertex v in G
$Key(v)$	The keyword set of vertex v
$\delta(G)$	$min\{deg_G(v)\|v \in V\}$
$G[H]$	The subgraph of G induced by a set of vertex H.
$J(S_1, S_2)$	The $Jaccard\ index$ between set S_1 and set S_2
G'	A subgraph of G s.t. $q \in G'$
$minDeg(C)$	The minimum degree of vertices in community C

Attribute-aware community (AAC). Conceptually, an AAC is a subgraph, G_0, of the attributed graph G satisfying:

1) **Connectivity.** G_0 is connected;
2) **Structure cohesiveness.** all the vertices in G_0 are tightly connected;
3) **Keyword cohesiveness.** all the vertices in G_0 have similar keywords set with each other.

Structure cohesiveness. One well known structure cohesiveness measure is the *minimum degree* of all the vertices that appear in the community has to be at least k [18] [30] [31] [19] [27] [32]. This is used in the k-core [18] [31] and our AAC.

Keyword cohesiveness. We observe the Jaccard index, which can measure the similarity between finite sample sets. If $J(Key(v), Key(q))$ is higher than a certain threshold, we can think that they have a high probability of being in the same community. We use this intuition to propose better algorithms. To ensure high Keyword cohesiveness, the Jaccard index between q and all the vertices of an AAC will be required to be at least r. The Jaccard index and ACS are defined as follows.

Definition 1. *(Jaccard index [33]) The Jaccard coefficient measures similarity between finite sample sets, and is defined as the size of the intersection divided by the size of the union of the sample sets:*

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Problem 1. *(ACS) Given a graph $G(V, E)$, a positive integer k, a positive number $r(0 \leq r \leq 1)$, a vertex $q \in V$ and a set of keywords $S \subset Key(q)$, return a subgraph $G_q \subset G$, and the following properties hold:*

1) ***Connectivity.*** *The subgraph G_q is connected and contains q;*
2) ***Structure cohesiveness.*** *$\forall v \in G_q,\ deg_{G_q}(v) \geq k$;*
3) ***Keyword cohesiveness.*** *$\forall v \in G_q,\ J(Key(v), S) \geq r$.*

Example 1. *Let us use the graph shown in Figure 1 as an example. Suppose the query vertex is $q(q = Jack, k = 3, r = 0.5, S = \{rock, blues\})$. In ACS, the subgraph G' induced by $H = \{Jack, Bob, Mike, John\}$ is the final solution because $\delta(G') = 3$ and $J(S, Key(v)) \geq 0.5, v \in V$. If $k = 2$, then there are multiple choices for H. For example, $\{John, Bob, Jack\}$, $\{Bob, Mike, Jack\}$ or $\{John, Mike, Jack\}$.*

We illustrate ACS in Example 1. As shown in the example above, ACS may return multiple answers. Here, we only focus on finding one solution that meets the AAC definition. Alternatively, we may want to find the smallest attribute-aware community. This is a new problem and we denote it as minACS.

Problem 2. *(minACS) Given a graph $G(V, E)$, a constant k, a positive number $r(0 \leq r \leq 1)$, a vertex $q \in V$ and a set of keywords $S \subset Key(q)$, find a subgraph G_q of G such that:*

1) The subgraph G_q is connected and contains q;
2) $\forall v \in G_q,\ deg_{G_q}(v) \geq k$;
3) $\forall v \in G_q,\ J(Key(v), S) \geq r$;
4) The size of G_q is minimized.

Unfortunately, the minACS problem is NP-complete. We omit its proofs due to space limitations, and the proof is reported in my github[1]. In this paper, we mainly focus on ACS problem.

4. ACS PROBLEM SOLUTION

In this section, we focus on developing ACS algorithms for community search in attributed graphs. Global search or establishment of auxiliary structures is costly and time-consuming as it needs to explore the entire graph. This is unacceptable when the graph is gigantic and dynamic. Our solutions follow the local search [19] framework. Intuitively, the best community for the query vertex is located near the vertex. Therefore, it is not necessary to retrieve the entire graph. The local search strategy works as follows: Initially, we set the query vertex as the target community G_q.The community expands as we explore in the vicinity of the query vertex and add eligible vertices into it. We stop the search when the current community satisfies $\delta(G_q) \geq k$.

4.1. The Baseline Solution

Consider the search process starting from the query vertex q. Each time we add one vertex into the target community until we get a solution H. Let $v_0, v_1, ..., v_t$ be the sequence of vertices which leads to the result. But as the community becomes larger, its minimum degree doesn't always become smaller. In other words, the minimum degree measure is not monotonic. This is demonstrated by Example 2.

1. https://github.com/jiahaochang/uic2018-NP-complete-proof

Example 2. *(Non-monotonicity). Suppose we want to find an attribute-aware community for $Jack$ in Figure 1. Assume the current community contains Jack's neighbors John and Mike. The subgraph induced by $H = \{Jack, Mike, John\}$ has a minimal degree of 2. To enlarge the community, we can add Bob or Alice. Adding Alice will decrease the minimal degree to 1, while adding Bob will increase it to 3.*

As shown in example 2, when we add a vertex to the current community, the new community does not necessarily have a bigger or smaller minimum degree. Thus, it is difficult for us to decide when to stop the search procedure. In [19], Cui et al. proposed an efficient algorithm $Local$, which proves that there always exists an order of vertices that monotonically leads to a solution. Let us discuss the theory first.

Theorem 1. *[19] For any vertex $v_0 \in H$ in graph G, there always exists a vertex sequence $v_0, v_1, ..., v_t$ of H starting with v_0 such that $\forall 0 \leq i < t, \delta(H_i) \leq \delta(H_{i+1})$.*

Algorithm 1: Basic Algorithm

```
Input: G(V, E), q, k, S, r
Output: H
H ← ∅;
H' ← {q} // the solution found so far
Function Search()
    if δ(G[H']) = k then
        H ← H';
        return;
    for vertex v in the neighbors of H' do
        if δ(G[H' ∪ {v}]) ≥ δ(G[H']) and J(Key(v), S) ≥ r then
            H' ← H' ∪ {v};
            Search();
            if H ≠ ∅ then
                return;
```

According to Theorem 1, we can pose a basic algorithm which is outlined in Algorithm 1. The function starts from $H' = \{q\}$. It exhaustively enumerates each vertices that is adjacent to H'. If the current vertex v satisfies both $\delta(H' \cup \{v\}) \geq \delta(H')$ and $J(Key(v), S) \geq r$, we add it to the target community. When a solution is found, the process will stop. Otherwise, it calls search recursively. From theorem 1 we can know that this exhaustive enumeration method can always find a valid solution to ACS. Obviously, the complexity of this baseline solution is exponential. Thus, we need to develop a more efficient ACS solution.

4.2. The Optimization Solution

In this section, we introduce an optimized search framework for ACS, which is outlined in algorithm 2. On the high level, it follows the *three-step* framework: (1) We assume that all the vertices have similar contexts or backgrounds with the query vertex q, and then we check if the graph meets the necessary condition of containing a subgraph that satisfies $\delta(G') \geq k$. (2) We start from the query vertex q, and at each step we select the local optimal vertex and add it into the current result set. In most cases, the second step will find a solution to the ACS problem. Otherwise, in step (3), we use global search strategy to find the solution from the subgraph induced by A. $G[A]$ consists of candidate nodes and is usually quite close to the solution, so this step is very fast.

Algorithm 2: Optimized Algorithm

```
Input: G(V, E), q, k, S, r
Output: H
H' ← {q};    // the solution found so far
A ← {q};  // vertices we have visited
B ← {v|(v, q) ∈ E, J(Key(v), S) ≥ r};
  // vertices we need to visit
if k > upperBound(G) then
    return
while B ≠ ∅ do
    Let v be the vertex with most links to H' from B;
    B ← B − {v_0};
    A ← A ∪ {v_0};
    if δ(G[A]) ≥ δ(G[H']) then
        H' ← A;
        if δ(G[H']) = k then
            H ← H';
            return
    Add v's neighbors with J(Key(v), S) ≥ r and degree larger than δ(G[H']) into B ;
if no solution is found then
    perform global search to find solution from subgraph G[A];
```

In the following sections, we first introduce the upper bound in Section 1). We present an intelligent candidate vertex selection strategy in Section 2), and in section 3) we give a complexity analysis of the algorithm.

4.2.1. Upper Bound. Sometimes there is no solution for ACS with respect to the query conditions. In order to avoid unnecessary work, we can tell if a graph G has a qualified solution before we perform search. Obviously, if the degree of the query node q is less than k, we can know there is no solution in graph G directly. In this section, we give an upper bound of $minDeg(C)$. If k is larger than the upper bound, then we can know that there is no solution.

Theorem 2. *[19] Given a connected simple graph $G(V, E)$, for a community C contained in $G(V, E)$, we have*

$$minDeg(C) \leq \lfloor \frac{1 + \sqrt{9 + 8(|E| - |V|)}}{2} \rfloor$$

4.2.2. Intelligent Candidate Vertex Selection Strategy. In this section, we present an effective search strategy. We start with the query vertex q and set it as the target community G_q. At each step, we select one candidate vertex from the vicinity of the current community. However, there may be more than one candidate vertices that satisfy the condition. If we blindly choose vertices from the candidate set, the following example shows that it may take more steps to find a solution.

Step1:C={Mike}	B={John, Jack, Bob, Alice}
Step2:C={Mike, Bob}	B={John, Alice, Jack}
Step3:C={Mike, Bob, Alice}	B={John, Jack}
Step4:C={Mike, Bob, Alice, John}	B={Jack}
Step5:C={Mike, Bob, Alice, John, Jack}	B={}

(a) blind selection

Step1:C={Mike}	B={John, Jack, Bob, Alice}
Step2:C={Mike, Bob}	B={John, Alice, Jack}
Step3:C={Mike, Bob, John}	B={Alice, Jack}
Step4:C={Mike, Bob, John, Jack}	B={Alice}

(b) Intelligent selection

Figure 2: Blind vs. Intelligent vertex selection

Example 3. *(VERTICES SELECTION STRATEGY). Consider the attributed graph shown in Figure 1. For query vertex $q(q = Mike, k = 3, r = 0.5, S = \{rock, blues\})$, using the blind selection method, Alice may be added into C (as shown in the Step 3 of Figure 2(a)), which will result in no valid solution. On the other hand, if we always choose the vertex that has the largest number of connections to the current community C from the candidate set B (as shown in Figure 2(b)), we can find a valid solution within four steps. The procedures for these two selection strategies are shown in Figure 2, where B is the candidate set from which the vertex is selected.*

The basic idea of optimizing the selection of vertices is to select the most promising vertex, which makes the fastest increase of the mean degree of the target community. The priority of a vertex v is defined as

$$Prio(v) = deg_{G[C \cup \{v\}]}(v)$$

At each step, we choose the vertex with the most number of connections to the current community. In general, the minimum degree of a community will increase with the growth of its density, and our strategy can accelerate the density growth of the target community. Consequently, this strategy may allow us to find a solution in fewer steps.

Example 4. *As shown in Figure 3, it is quite possible that Alice is selected at step 1. Therefore, no solution can be found even after all the vertices have been visited.*

As shown in Example 3, this optimized selection strategy is generally quite efficient. However, Example 4 shows that

Step1:C={Mike}	B={John, Jack, Bob, Alice}
Step2:C={Mike, Alice}	B={John, Bob, Jack}
Step3:C={Mike, Alice, Bob}	B={John, Jack}
Step4:C={Mike, Alice, Bob, John}	B={Jack}
Step5:C={Mike, Alice, Bob, John, Jack}	B={}

Figure 3: No solution found by intelligent candidate generation strategy

sometimes local information is not enough to construct a valid solution. In this case, we need to perform a global search in G[A] (line 17) of Algorithm 2. This step is very fast. Because all the vertices in subgraph G[A] satisfy keyword cohesiveness and G[A] is usually very close to the final solution. This step ensures a valid solution if it exists.

4.2.3. Complexity. After our careful design, the optimized algorithm can be implemented in linear time. Let n and m represent the number of vertices and edges in G[A], respectively. The idea is to make a list of vertices with priority $Prio(v)$, for $Prio(v) = 1, ...$, that is, one separate list for each priority. When adding a vertex u to C, a neighbor of u (except those already in C) with priority $Prio(u)$ needs to be moved from the list $Prio(u)$ to the list $Prio(u) + 1$. In addition, we can locate a vertex with maximum priority for the next step in O(1) time. Overall, the running time is O(n + m).

5. EXPERIMENTS

We now present the experimental results of our algorithm. We discuss the setup in Section 5.1. Sections 5.2 and 5.3 report the results on real datasets.

5.1. Setup

Datasets. We consider the following three datasets: Facebook[2], DBLP[3] [34], Tencent Weibo[4]. For Facebook and Tencent Weibo, each vertex represents a person, and edges represent interactions between people. The features of each vertex are extracted from the user profile. For DBLP, each vertex denotes an author, and each edge represents a coauthor relationship. For each author, we use the research interests of this author as his keywords. The statistics of these graphs are showed in Table 2, where the vertex's average keywords set size is given as $Avg(key)$.

TABLE 2: **Real datasets used in our experiments**

Dataset	Vertices	Edges	Avg Degree	Avg(key)
Facebook	4039	88234	43.6	7.37
Tencen Weibo	2320895	50133369	43.2	6.96
DBLP	1712433	4258947	5.0	8.52

2. http://snap.stanford.edu/data/egonets-Facebook.html
3. https://www.aminer.cn/billboard/aminernetwork
4. https://www.kaggle.com/c/kddcup2012-track1

Solutions to ACS are sensitive to the input parameter k and r. To evaluate the performance of our ACS algorithms, we set 5 as the default value of k and 0.3 as the default value of r. The input keywords set $Key(q)$ is set as the whole set of keywords contained by the query vertex q. For each real dataset, we randomly selected 100 vertices as the query vertex with core numbers of 5 or more so that there was always a solution containing the query vertex. Then, we used the average query time of these 100 query vertices as result.

We implemented all of the algorithms in Java, and ran the experiments on a PC having Intel Core i7-6700 processor, and 16GB of memory, with Windows installed.

5.2. The effectiveness of our ACS method

We first define a measure to evaluate the keyword cohesiveness of the community.

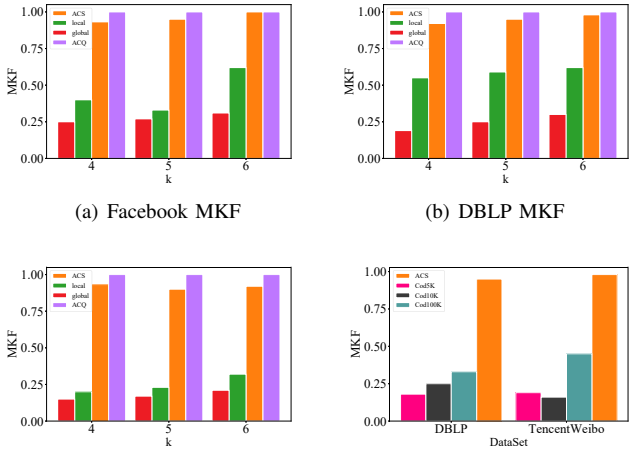
5.2.1. Maximum keyword frequency (MKF). Consider a keyword $x \in Key(q)$. If x appears in most of the nodes of a community C, we can regard C to be highly keyword cohesive. And x can effectively reflect the theme of this community. Let f_i be the number of vertices of C whose keyword sets contain the i-th keyword of $Key(q)$. Then, $\frac{f_i}{|C|}$ is the occurrence frequency of this keyword in C. The MKF is the max of this value over all keywords in Key(q).

$$MKF(C) = max\{\frac{f_i}{|C|} | i = 1, 2, ..., |Key(q)|\}$$

The $MKF(C)$ value has a range of 0 and 1. A higher value of $MKF(C)$ implies that the community has better keyword cohesiveness.

5.2.2. Comparison with the existing community search (CS) methods. Most of the existing CS methods are mainly designed for non-attributed graphs. We have implemented two state-of-the-art methods: *Global* [18] and *Local* [19]. Both of them use the *minimum degree* to measure the structure cohesiveness. And we also choose a representative index-based algorithm ACQ [20] as a comparison, which can be used in attributed graphs. Figures 4(a)-4(c) show the MKF values for the three datasets. We can see that the keyword cohesiveness of ACS is higher than both *Global* and *Local*, because ACS considers vertex keywords, while *Global* and *Local* do not. With the help of CL-tree, the MKF of ACQ is slightly higher than our algorithm. But the index creation process is very time-consuming, especially when the graph structure is constantly changing. We will discuss this in the following.

5.2.3. Comparison with the existing community detection (CD) methods. As mentioned ahead, there exists CD methods that can be used for attributed graphs. We choose CODICIL [23] for comparison. The main reasons are: (1) it identifies communities of better quality than those of many existing methods; and (2) It runs relatively fast. Since CODICIL needs users to specify the number of clusters expected, we set the numbers as 5K, 10K, and 100K. The corresponding adapted algorithms are named as Cod5K, Cod10K, Cod100K respectively. We first run these algorithms offline to obtain all the communities, which takes 3+ days. Given a query vertex q, they return communities containing q as the results. Figure 4(d) shows that ACS always performs better than CODICIL, in terms of MKF.



(a) Facebook MKF (b) DBLP MKF

(c) TencentWeibo MKF (d) Comparing with CD method

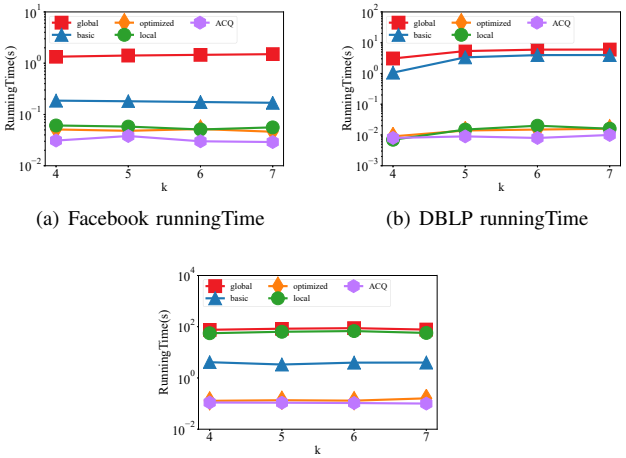
Fig. 4: Comparing with existing methods.

5.3. The efficiency of our ACS method

5.3.1. Effect of k. In this section, we evaluate the efficiency of our ACS method by comparing it with *Global* and *Local*. Both of them can do community search without the support of index structures. And we choose the index-based algorithm ACQ [20] as a comparison. The reasons are as follows: (1) It runs faster than many existing index-based methods like [10] [9]. (2) It finds communities of relatively high quality.

Figures 5(a)-5(c) compare the query efficiency under different k. Note that our ACS method performs faster than *Global* method. *Optimized-ACS* method performs better than *basic-ACS* and *Local* for most cases. For the largest dataset Tencent weibo, optimized-ACS method is at least two orders of magnitude faster than *Local*. This is because ACS finds a relative smaller community than *Local*. If we do not take into account the time consumption of index construction, the ACQ is faster than our optimized method. However, the querying time for ACQ and optimized-ACS is in the same time magnitude. Thus, optimized-ACS is as efficient as ACQ.

5.3.2. Effect of the change of graph structure. Figures 6(a)-6(c) compare the efficiency of ACQ and our optimized method in dynamic networks. For each dataset, we randomly select 20%, 40%, 60% and 80% of its vertices, and obtain four subgraphs induced by these vertex sets. We use them to

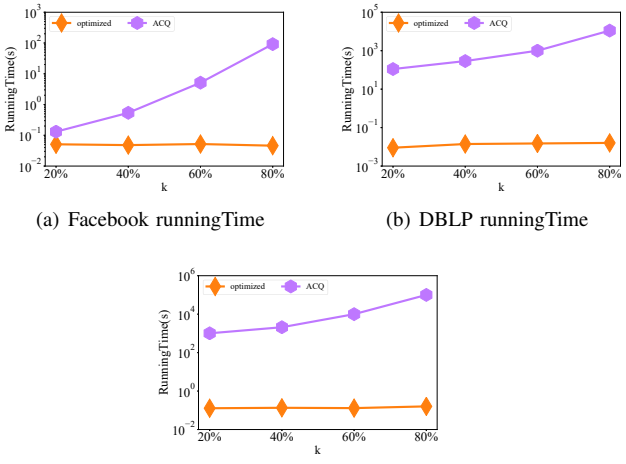


(a) Facebook runningTime (b) DBLP runningTime

(c) TencentWeibo runningTime

Fig. 5: Comparing with CS methods.

simulate the graph structure at different times. In this case, we have to update the index for ACQ every time the graph structure changes. Therefore, the ACQ query time consists of two parts, which are the time cost of building the index and the time cost of querying according to the index. We can see from the results that the performance of ACQ becomes unacceptable, while our method can still quickly get the query results.

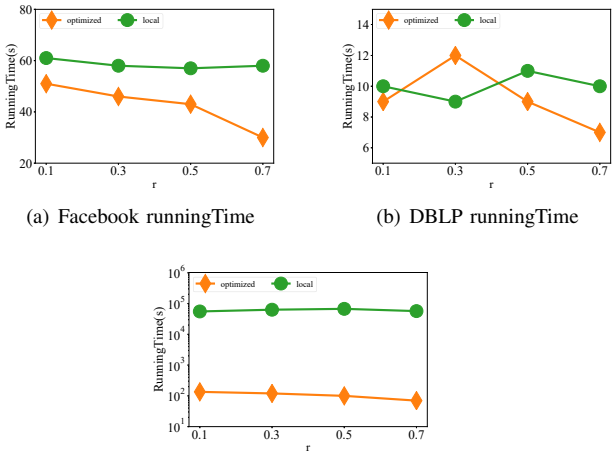


(a) Facebook runningTime (b) DBLP runningTime

(c) TencentWeibo runningTime

Fig. 6: The running time with different graph structure

5.3.3. Effect of r. In this section, we discuss the CS methods (*Global*, *Local* and our *ACS*) without index assistance, which can be applied to dynamic networks. For each query vertex, we randomly select 0.1, 0.3, 0.5, 0.7 as the value of r. As optimized-ACS performs better than basic-ACS and *Global*, we mainly compare optimized-ACS with *Local* method. Figures 7(a)-7(c) show that the cost of optimized-ACS algorithm decreases with the r. This is because the communities found by ACS become smaller as r increases.



(a) Facebook runningTime (b) DBLP runningTime

(c) TencentWeibo runningTime

Fig. 7: The running time with different r

6. Conclusion

An AAC is a community that satisfies both structure and keyword cohesiveness. We investigate the problem of finding a meaningful community that contains the query vertex. Our method needn't the support of index structures and can be used in dynamic networks. The experimental results show that the performance of our ACS in dynamic graphs is still pretty good. Our future work will consider the overlapping communities in attributed graphs. Overlapping community structures are vital in revealing the internal structure of large networks.

ACKNOWLEDGMENTS

This work is partially supported by National Natural Science Foundation of China (NSFC) under Grant No.61772491, No.61472460, and Natural Science Foundation of Jiangsu Province under Grant No. BK20161256. Kai Han is the corresponding author.

References

[1] Z. Xu, Y. Ke, Y. Wang, H. Cheng, and J. Cheng, "A model-based approach to attributed graph clustering," in *ACM SIGMOD International Conference on Management of Data*, 2012, pp. 505–516.

[2] B. Ding, J. X. Yu, S. Wang, and L. Qin, "Finding top-k min-cost connected trees in databases," in *IEEE International Conference on Data Engineering*, 2007, pp. 836–845.

[3] H. He, H. Wang, J. Yang, and P. S. Yu, "Blinks: ranked keyword searches on graphs," in *ACM SIGMOD International Conference on Management of Data*, 2007, pp. 305–316.

[4] V. Kacholia, S. Pandit, S. Chakrabarti, S. Sudarshan, R. Desai, and H. Karambelkar, "Bidirectional expansion for keyword search on graph databases," in *International Conference on Very Large Data Bases, Trondheim, Norway, August 30 - September*, 2014, pp. 505–516.

[5] J. Yu, L. Qin, and L. Chang, "Keyword search in databases," *IEEE Data Eng Bull*, vol. 1, no. 1, pp. 681–694, 2009.

[6] M. Kargar and A. An, "Keyword search in graphs: finding r-cliques," *Proceedings of the Vldb Endowment*, vol. 4, no. 10, pp. 681–692, 2011.

[7] Y. Fang, H. Zhang, Y. Ye, and X. Li, "Detecting hot topics from twitter: A multiview approach," *Journal of Information Science*, vol. 40, no. 5, pp. 578–593, 2014.

[8] J. Yang, J. Mcauley, and J. Leskovec, "Community detection in networks with node attributes," pp. 1151–1156, 2014.

[9] L. V. S. Lakshmanan and L. V. S. Lakshmanan, *Attribute-driven community search*. VLDB Endowment, 2017.

[10] J. Shang, C. Wang, C. Wang, G. Guo, and J. Qian, "An attribute-based community search method with graph refining," *Journal of Supercomputing*, pp. 1–28, 2017.

[11] B. Perozzi and L. Akoglu, "Focused clustering and outlier detection in large attributed graphs," in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2014, pp. 1346–1355.

[12] J. Zheng, H. Xu, G. Chen, H. Dai, and J. Wu, "Congestion-minimizing network update in data centers," *IEEE Transactions on Services Computing*, vol. PP, no. 99, pp. 1–1, 1939.

[13] J. Zheng, G. Chen, S. Schmid, H. Dai, J. Wu, and Q. Ni, "Scheduling congestion- and loop-free network update in timed sdns," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2542–2552, 2017.

[14] Y. Qu, C. Dong, H. Dai, F. Wu, S. Tang, H. Wang, and C. Tian, "Multicast in multihop crns under uncertain spectrum availability: A network coding approach," *IEEE/ACM Transactions on Networking*, vol. PP, no. 99, pp. 1–14, 2017.

[15] H. Dai, Y. Zhong, A. X. Liu, W. Wang, and M. Li, "Noisy bloom filters for multi-set membership testing," in *ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science*, 2016, pp. 139–151.

[16] H. Dai, G. Chen, C. Wang, S. Wang, X. Wu, and F. Wu, "Quality of energy provisioning for wireless power transfer," *IEEE Transactions on Parallel & Distributed Systems*, vol. 26, no. 2, pp. 527–537, 2015.

[17] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, no. 3–5, pp. 75–174, 2010.

[18] M. Sozio and A. Gionis, "The community-search problem and how to plan a successful cocktail party," in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, Dc, Usa, July*, 2010, pp. 939–948.

[19] W. Cui, Y. Xiao, H. Wang, and W. Wang, *Local search of communities in large graphs*. ACM, 2014.

[20] Y. Fang, R. Cheng, S. Luo, and J. Hu, "Effective community search for large attributed graphs," *Proceedings of the Vldb Endowment*, vol. 9, no. 12, pp. 1233–1244, 2016.

[21] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks." *Physical Review E*, vol. 69, no. 2 Pt 2, p. 026113, 2004.

[22] Y. Zhou, H. Cheng, and J. X. Yu, *Graph clustering based on structural/attribute similarities*. VLDB Endowment, 2009.

[23] Y. Ruan, D. Fuhry, and S. Parthasarathy, "Efficient community detection in large networks using content and links," pp. 1089–1098, 2012.

[24] X. Huang, H. Cheng, and J. X. Yu, *Dense community detection in multi-valued attributed networks*. Elsevier Science Inc., 2015.

[25] H. Cheng, "Clustering large attributed information networks: an efficient incremental computing approach," *Data Mining and Knowledge Discovery*, vol. 25, no. 3, pp. 450–477, 2012.

[26] W. Cui, Y. Xiao, H. Wang, Y. Lu, and W. Wang, "Online search of overlapping communities," in *ACM SIGMOD International Conference on Management of Data*, 2013, pp. 277–288.

[27] R. H. Li, L. Qin, J. X. Yu, and R. Mao, "Influential community search in large networks," *Proceedings of the Vldb Endowment*, vol. 8, no. 5, pp. 509–520, 2015.

[28] X. Huang, L. V. S. Lakshmanan, J. X. Yu, and H. Cheng, "Approximate closest community search in networks," *Proceedings of the Vldb Endowment*, vol. 9, no. 4, pp. 276–287, 2015.

[29] J. Hu, X. Wu, R. Cheng, S. Luo, and Y. Fang, "Querying minimal steiner maximum-connected subgraphs in large graphs," pp. 1241–1250, 2016.

[30] S. B. Seidman, "Network structure and minimum degree," *Social Networks*, vol. 5, no. 3, pp. 269–287, 1983.

[31] V. Batagelj and M. Zaversnik, "An o(m) algorithm for cores decomposition of networks," *Computer Science*, vol. 1, no. 6, pp. 34–37, 2003.

[32] Y. Fang, R. Cheng, X. Li, S. Luo, and J. Hu, "Effective community search over large spatial graphs," *Proceedings of the Vldb Endowment*, pp. 709–720, 2017.

[33] P. Jaccard, "Etude de la distribution florale dans une portion des alpes et du jura," *Bulletin De La Societe Vaudoise Des Sciences Naturelles*, vol. 37, no. 142, pp. 547–579, 1901.

[34] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su, "Arnetminer:extraction and mining of academic social networks," in *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2008, pp. 990–998.