A Progressive Approach for Neighboring Geosocial Communities Search Over Large Spatial Graphs

ZEWEN WU, JIAN XU, HUAIXIANG ZHANG, QING BAO, QINGSUN, AND CHANGBENGZHOU
School of Computer, Hangzhou Dianzi University, Hangzhou 310018, China
Corresponding author: Zewen Wu (zewenwu@hdu.edu.cn)
This work was supported by the National Natural Science Foundation of China under Grant 61702391, Grant U1809206, and Grant 61806061.

ABSTRACT Searching for neighbors for a query node in a spatial network is a fundamental problem and has been extensively investigated. However, most existing works focus only on the node level when conducting such a query and rarely pay attention to the social relations among the neighbors. We argue that a user, in some cases, is more likely to engage in some activities collectively, i.e., going to the bar with friends rather than alone. For this reason, we consider the neighbor searching problem at a community level in this paper and examine a new problem: Neighboring Geosocial Communities Search (NGCS) over large spatial graphs. Specifically, given a parameter $n$ and query node $q$, we aim to find the top-$n$ nearest communities for $q$. Moreover, in each returned community, nodes have cohesive relations with each other and are covered by a minimum covering circle (MCC) whose radius is less than $r$. It is obvious that the NGCS problem finds its standard applications in marketing and other scenarios but it is very challenging for large spatial graphs because it requires detecting all qualified cohesive user communities. Therefore, in this paper, we adopt a local search approach to reduce the difficulty. The introduced algorithm finds the top-$n$ neighboring geo-social communities through a progressive search in the graph without thoroughly examining the graph. Analyses show that the complexity of the algorithm is decreased by an order of magnitude. Extensive experiments on real social networks confirm the superiority and effectiveness of our solutions.

INDEX TERMS Social network services, nearest neighbor searches, spatial network.

I. INTRODUCTION Community structures are ubiquitous in numerous real-world networks, such as social, collaboration and communication networks [1]. Different from community detection, which identifies all communities in a graph, the community search problem involves finding densely connected communities that satisfy a user-specified query [2].

The proliferation of smartphones and other GPS-enabled smart devices has led to the rapid growth of location-based social networks (LBSNs), which are also known as geosocial networks. Popular geosocial networks include Facebook, Foursquare, Google+ etc. For example, Foursquare hosts data on more than 105 million global points of interest and over 500 million user devices. Similar to general social networks, there is also a great need to interrogate the global organization of such networks in terms of their structural subunits [3]. Therefore, the problem of searching for geosocial groups in geosocial networks has recently been proposed and studied by [4]–[6], which considers both the community structural cohesiveness and the spatial proximity.

In this work, we focus on finding top-$n$ communities around a given query user or location. The communities we intend to obtain in the query are densely connected and spatially close. To the best of our knowledge, this is the first work to find communities around a query vertex that satisfies the given spatial constraint.

Figure 1 depicts a geosocial graph example with ten users. The social layer illustrates the social relationship among the users, and the spatial layer shows the location information of each user.

We take common dense subgraph definitions of the $k$-core to measure structural cohesiveness in this work. The $k$-core of a graph $G$ is the largest subgraph of $G$ in which every vertex
is adjacent to at least $k$ other vertices within the subgraph [7]. Take Figure 1 as an example, and let $v_{10}$ be the query vertex, with $k = 3$. Obviously, communities formed by vertices $\{v_1, v_2, v_5, v_6\}$, $\{v_2, v_3, v_6, v_7\}$ and $\{v_4, v_5, v_8, v_9\}$ are three qualified 3-core user groups consisting of 4 vertices around vertex $v_{10}$. However, note that only $\{v_2, v_3, v_6, v_7\}$ and $\{v_4, v_5, v_8, v_9\}$ fall in a circle with a radius less than $r$; then, the output only includes two as the results, which satisfy both structural cohesiveness ($k = 3$) and spatial proximity (covered by a circle of radius less than $r$). Specifically, in this paper, we propose the Neighboring Geosocial Communities Search (NGCS), given a large spatial graph and a query node, which aims to find the top-$n$ nearest subgraphs around the query node such that the users in the subgraph are socially cohesive and spatially close.

The NGCS has many real-life applications. Two representative applications are discussed in the following.

Social marketing: People with close social relationships and spatial proximity may shop or patronize places that are also physically near. We figure out that the chances of success of the social activity will be higher if they invite a set of friends so that each participant is well acquainted with the others. Some online review services, such as Yelp, allow users to explore and search restaurants, bars and other places. For restaurant owners to promote sales, advertisements can be sent to the geosocial groups of users who might visit restaurants. NGCS can be used to find communities of users who have close relations and live together. They will be the ideal target customers since they are more likely to eat out at restaurants together. The closer they live from the restaurant, the more likely they will be patronized.

Geosocial Data Analysis: Studying the features of geographical regions is an important problem in data analysis. With NGCS, we can analyse the characteristics of communities around a specific location. For instance, suppose we have a region that has team sports facilities such as a basketball court. It is useful to know the basketball lovers living nearby and those who like to play together (gangs with close relationships). By investigating the distribution of gangs around the current court, it is easy to determine the demand and the potential locations for a possible new basketball court.

However, the NGCS problem is very challenging because the area in the graph that needs to be searched is unknown. A straightforward approach would require an algorithm to detect all the qualified cohesive user communities first and then examine all these communities to see if each of them is covered by a fixed size circle. To go over an entire graph, the cost is obviously very high and is impractical for large spatial graphs with millions of vertices. Therefore, in this paper, we turn to a local search approach, which is able to return results with less cost. We also propose algorithms adopting a local search approach that compute the top-$n$ neighboring geosocial community by conducting a progressive search on graph $G$ without thoroughly examining the graph to overcome the deficiencies of the straightforward algorithm.

The contributions of our work can be summarized as follows:

- We raise the novel Neighboring Geosocial Community Search (NGCS) problem for the first time.
- A progressive local search approach is introduced to conduct NGCS, and an algorithm that implements the search approach is also devised and the details are analysed in the paper.
- We also propose an approximation algorithm to improve the performance and efficiency in solving the problem.
- The results of extensive experiments on real social networks show the correctness of our algorithms and demonstrate that algorithms adopting a progressive local search approach significantly outperform the baseline algorithm.

The remainder of this paper is organized as follows. First, we introduce the related work in Section II. Then, we propose and formulate the NGCS problem in geosocial networks and present a basic solution in Section III. A progressive local search algorithm is proposed in Section IV. In Section V, we present an approximation algorithm to improve the searching performance and efficiency. The experimental results are illustrated in Section VI. Finally, we conclude this work in Section VII.

II. RELATED WORK

A. SOCIAL COMMUNITY COHESIVENESS

Social community cohesiveness is a fundamental concept in graph structure analysis [8]. A cohesive community is a set of nodes that has more edges between its members than with the remainder of the network. Cohesive community definitions have been proposed by several existing works and can be categorized as: (1) clique and quasi-clique [8]–[10], (2) k-truss [5], [11]–[16], (3) k-core [17]–[20], and so on.

A $k$-clique community is a complete subgraph of $k$ vertices, where every pair of vertices is adjacent. It is the densest graph among all the $k$-node graphs. The $k$-truss community of a graph is a set of maximal connected subgraphs and requires that every edge in the community is contained in at least $(k-2)$ triangles. However, both of them have strict cohesiveness requirements, which makes it impractical to apply in large communities in real-world networks. Recently, the k-core model has received much interest for its elegant structures as well as efficient solutions. The $k$-core of a graph is a set of maximal connected subgraphs and requires that
every vertex in the subgraph has at least \( k \) neighbors. The \( k \)-core model has several desirable properties: cohesiveness, maximum, connectivity, and efficiency. Specifically, all vertices in a \( k \)-core community have a degree of at least \( k \) (cohesiveness); a \( k \)-core community is not a subgraph of another \( k \)-core community (maximum); any two vertices in the \( k \)-core community are connected; and \( k \)-core decomposition is an efficient process in contrast to \( k \)-cliques and \( k \)-truss computing. For the reasons above, we use the \( k \)-core as our community pattern.

### B. SOCIAL COMMUNITY SEARCH

Different from community detection, the purpose of community search is to find communities containing a given query node \( q \). Social community search is a long-studied problem that was first introduced in [19]. It is different from social community detection since the query nodes \( q \) are given. Several factors should be taken into consideration when searching the community [21], these are efficiency, scalability, quality, personalization, and support for dynamic graphs. Specifically, community search algorithms should be able to respond in real time (efficiency) and adapt to real-world big datasets (scalability). The community search results should be cohesively connected (quality). Users are usually interested in a personalized community search, which returns different results with different query nodes \( q \) (personalization). Because a graph in the real world is usually dynamic, searching algorithms should adapt to dynamic changes easily (support for dynamic graphs). Based on the \( k \)-core model, some recent works mainly focus on other attribute information. For example, Zhang et al. [22] propose a novel cohesive community model, named the \( (k, p) \)-core, which helps refine the \( k \)-core model to more accurately capture the engagement dynamics of users. Fang et al. [23] consider both keyword and structure cohesiveness when searching communities. Li et al. [24], [25] and [26], [27] studied the influential community search problem. Zhang et al. [28] proposed a novel cohesive subgraph model for attributed graphs, called the \((k, r)\)-core, to capture the cohesiveness of subgraphs from both the graph structure and the vertex attributes.

### C. GEOSOCIAL COMMUNITY SEARCH

A geo-social network contains communities that are spatially proximate. Most community search algorithms introduced in previous studies do not consider the vertices’ spatial information. For this reason, spatial information is considered an attribute of nodes in community search [4], [6], [29]–[32]. A GeoSocial community is defined as a community in which vertices are structurally and spatially cohesive. Fang et al. [4] introduced the spatial-aware community (SAC), whose vertices are close structurally and spatially. Wang et al. [29] used the \( k \)-core to ensure social cohesiveness, and they used a radius-bounded circle to restrict the locations of users. Al-Baghdadi et al. [30] retrieved communities with high social influence and small traveling times and covered certain keywords. Li et al. [31] proposed a new approach that considers the constraints equally and studies a skyline query. It helps users decide which constraints they need to choose and how to set the priority of the constraints to meet their real requirements. Zhong et al. [32] proposed a height-balanced and scalable index, namely, G-tree, to efficiently support various queries within one framework. Index-based methods can efficiently retrieve communities from a prebuilt index. However, index building and index updating are time consuming when graphs change, so they are not suitable for dynamic graphs. Kim et al. [6] aim to find a social community and a cluster of spatial locations that are densely connected in a location-based social network simultaneously.

However, the algorithms proposed in previous studies on community search cannot be directly used for our problem, which is different from these existing community searches. To the best of our knowledge, our work is the first to study the Neighboring Geosocial Communities Search (NGCS) problem.

Our work is fundamentally different from SAC [4] in at least two aspects. First, the problem definitions are different. Given a query node \( q \), the SAC problem aims to retrieve the community containing the node \( q \). Whereas in our proposed NGCS problem, we aim to retrieve the neighborhoods of \( q \) (which may not contain a point \( q \)). Different from the SAC problem, the NGCS problem not only considers structural cohesiveness and location closeness but we also impose constraints on the distance between the community and the query node \( q \). Second, the application scenarios are different. The SAC problem is simply to retrieve only one community with structural cohesion and location closeness, and the application scenario is to help a single user group organize gatherings. However, the NGCS considers problems from the perspective of merchants, and helps merchants find the top-\( n \) communities in order of distance, which is convenient for merchants when they advertise.

### III. PROBLEM AND BASIC SOLUTION

In this section, we formally define our NGCS problem over large spatial graphs. A basic method follows the local search approach. Table 1 summarizes the mathematical notations used throughout this paper.

### A. PRELIMINARIES

We model a location-based social network with a spatial graph \( G(V, E) \), which is an undirected graph with a vertex

| Table 1. Summary of notations. |
|-------------------------------|
| Symbol | Description |
| \( G(V, E) \) | A graph, node set \( V \), edge set \( E \) |
| \(|V|, |E|\) | The size of \( V \) and \( E \) |
| \( G(V_s) \) | An induced subgraph of \( G \) |
| \( \deg G(v) \) | The degree of vertex \( v \) in \( G \) |
| \( \beta \) | Introduced error |
| \( \varepsilon \) | True error |
set V and edge set E, where the vertices represent users in the network and the edges denote their relationships. |V| and |E| are the corresponding numbers of vertices in V and edges in E. For each vertex v ∈ V, we denote its position with location (v.x, v.y), where v.x denotes its longitude and v.y denotes its latitude in a two-dimensional space. We calculate the distance between vertices u and v by using their longitude and latitude.

Definition 1: (Induced Subgraph)
A graph G(V_S, E_S) is called a subgraph of G induced by vertex set V_S, where (1) V_S ⊆ V, (2) E_S ⊆ E and (3) for every edge (u, v) ∈ E_S, u, v ∈ V_S.

A widely used notion of structure cohesiveness is that the minimum degree of all the vertices that appear in the community is at least k. This is a well-known k-core [7] and is also used in this work. It is formally defined as follows:

Definition 2: (k-Core [7])
Given an integer k (k ≥ 0), the k-core of a graph G, denoted by G^k, is the largest induced subgraph of G, such that ∀v ∈ G^k, degG^k(v) ≥ k.

Note, we have that G^{k+1} ⊆ G^k or G^k has an order of k [33].

The core number of a vertex v ∈ V is then defined as the highest order of the k-core that contains v. A k-core then has some important properties: (1) k-cores are nested; (2) cores are not necessarily connected subgraphs; (3) G^k contains at least k + 1 vertices; and (4) the total time complexity of the k-core decomposition algorithm is O(max(m, n)), or it can be completed using a linear algorithm [33].

When using the k-core to measure the cohesiveness of the user community in spatial graphs, it is important to require that any returned k-core community be a connected subgraph. Similar to many k-core search algorithms [4], [18], we also use k-core to denote k-core community-constituted connected components.

Definition 3: (Geosocial Communities)
Given a positive integer k, a Geosocial Communities (GC) in a spatial graph G is an induced subgraph G(V_S) from G and has the following properties:
1. Connectivity. G(V_S) is connected;
2. Structure cohesiveness. ∀v ∈ V_S, degG_S(v) ≥ k;
3. Spatial closeness. ∀v ∈ V_S, covered by a minimum covering circle (MCC), and its radius is less than the predefined r.

Although we use k-core as the structure cohesiveness metric, it is important to notice that algorithms introduced in this paper can be easily adapted to work with other criteria like k-truss [2] and k-clique [9]. To be precise, all the methods in this paper do not limit the structural cohesiveness metric. For example, if we set k-truss as the metric, we only need to change the k-core computation method in the algorithm to the k-truss computation method. To ensure spatial closeness, we require all the vertices in a community to reside in a fixed size circle. This is a notion to achieve high spatial closeness for a set of spatial objects, in concert with many previous studies [34].

Definition 4: (Neighboring Geosocial Communities Search)
Given a vertex q ∈ V and a positive number n, the NGCS returns a sequence of geosocial communities < GC1, GC2, GC3, …, GCn > where GC_i ∈ G (1 ≤ i ≤ n), and the order of the sequence is decided by their distances from the farthest vertex in each community to the query vertex q in ascending order.

NGCS is actually a top-n query for vertex q. The intuition of NGCS is as follows. Given a query node q, a positive number n, minimum degree k and radius r, the algorithm searches for a sequence of nearest user communities, each of which satisfies all three properties in Definition 3. These n communities are ordered by the farthest distance to vertex q.

The intuition behind our definition is that each node in a community should have a short distance to query node q, indicating that every member in the community can easily reach node q with a low cost. Therefore, in this paper, we use the distance between the farthest vertex in the community and the query vertex to define the distance between a community and the query vertex.

Taking the geosocial graph in Figure 1 as an example, there are two qualified communities \{v_2, v_3, v_6, v_7\} and \{v_4, v_5, v_8, v_9\}, which are connected and structurally and spatially cohesive. Therefore, these two communities are the query results. Since we obtain more than one such community in this figure, we need to order them by Definition 4. Suppose \{v_4, v_5\} and \{v_3\} are the farthest nodes and \{v_4, v_5\} is closer to query node q than \{v_3\}. Then, GC \{v_4, v_5, v_6, v_9\} will be the top-1 nearest GC, and the other is top-2.

B. BASIC SOLUTION
We now present our basic algorithms that adopt the local search approach. The solutions follow the two-step framework: (1) the first core decomposition of the graph, which obtains all communities based on some algorithms, e.g., Global [7], [19], and (2) examine all of the communities to obtain a subset that satisfies both structural and spatial cohesiveness and returns n communities that are nearest to a vertex q. It is easy to see that Step (2) is computationally challenging since there are exponential numbers of possible communities.

A naive approach is to enumerate all the possible communities of the subgraph around the vertex q and then choose the one that satisfies all three criteria of GC. In this paper, we propose to use an exponential growth strategy for enlarging the searched vertices; that is, we iteratively increase the number of vertices in the subgraph, with a growing ratio of α, for processing. In this way, we only need to work on a subgraph that can be much smaller than G. A sketch of the solution is presented in Algorithm 1.

We assume that the vertices of G are presorted in increasing order with respect to their distance to vertex q. Algorithm 1 first initializes i and d_i. In detail, d_i represents the minimum number of vertices for n eligible communities (Line 1). Then, it induces the subgraph G' from G by ListV[1..d_i] (Line 3).
Algorithm 1 NGCSBasic.

Input:
A graph $G$, sorted vertex list $ListV$, vertex $q$ and radius $r$

Output:
A $k$-core list $ListKC$
1: Initialize $i = 1, d_i$;
2: $V' = set(ListV[1..d_i])$;
3: Copy $G' = (V', E')$ from $G$;
4: while ($G' \neq G$) do
5:   Detect $k$-core in $G'$;
6:   Create vertex list $List \Lambda$ for $k$-core;
7:   Sort list $List \Lambda$;
8:   $ListKC = GCExaminate(q, List \Lambda, r)$;
9:   if $ListKC.length > n$ then
10:      return $ListKC$;
11:      $d_{i+1} = \alpha \ast d_i$;
12:      $i = i + 1$;
13:   $V' = set(ListV[1..d_i])$;
14:   Copy $G' = (V', E')$ from $G$;
15: return $ListKC$;

Algorithm 2 GCExaminate.

Input:
A vertex list $List \Lambda$, vertex $q$ and radius $r$

Output:
A $k$-core list $ListKC$
1: Initialize $ListKC$;
2: for $h = 3$ to $|List \Lambda|$ do
3:   for $i = 1$ to $h - 2$ do
4:     for $j = i+1$ to $h - 1$ do
5:       $circle = MCC(X_h, List \Lambda_i, List \Lambda_j)$;
6:       if $circle.radius < r$ then
7:          $S = \{v | v \text{ inside } circle\}$;
8:          $\Lambda = the \ k$-core in $G(V_S)$;
9:          if $\Lambda \neq \emptyset$ then
10:             Insert $\Lambda$ into $ListKC$;
11:         Sort list $ListKC$ by the furthest vertex to $q$;
12:      return $ListKC$;

In the while loop (Lines 4-14), it first detects $k$-core in $G'$ (Line 5). We know that computing the number of $k$-core in a graph $G$ can be done in linear time to the size of $V$ and the size of $E$ [33]. To determine the connectivity of the graph, we can use a depth-first search, breadth-first search or disjoint set algorithm. All of these algorithms can also be conducted in linear time. This means that the running time of Line 5 in Algorithm 1 is linear to the size of the problem. Next, it creates a vertex list $ListV$ for $k$-core and sorts the vertices of the eligible $k$-core in ascending order (Lines 6-7). To sort list $X$ in Line 7, it requires $O(|V'|log|V'|)$ comparisons in the worst case, where $|V'|$ equals the length of the list. Line 8 invokes a function GCExaminate() to check GCs in the subgraph. Then, as long as the subgraph $G'$ under consideration contains less than $n$ GCs and $G'$ is not equal to $G$ (Line 4), we start the next iteration of searching such that the vertices in the subgraph are times $V'$. Each time it increases the size of $V'$ exponentially, if $d_i$ is larger than $|ListV|$, we set $d_i = |ListV|$. Finally, if the size of the returned $ListKC$ is greater than $n$, Algorithm 1 will stop and return. If it is conclusive that there are fewer than $n$ GCs after a thorough search in graph $G$, Algorithm 1 also returns (Line 15).

NGCSBasic algorithm is inspired by the research of [18] and [27]. They all used the local search strategy, which searches in the neighborhood of a vertex to find the best community for the vertex. Based on this idea, we proposed the NGCSBasic algorithm and set it as our baseline algorithm.

It is known that $k$-core are nested [33]. The example shown in Figure 1 has nested $k$-core, where $\{v_2, v_3, v_6, v_7\}$ is $k$-core with $k=3$, which is nested in $\{v_1, v_2, v_3, v_5, v_6, v_7\}$. Note that when we examine some $k$-core covered by a larger minimum circle, which do not satisfy the required $r$, their subgraph $\{v_3, v_4, v_7, v_8\}$ may satisfy the third property of GC. Therefore, it is necessary to go over every possible circle in $k$-core.

J. Elzinga [34] posits that given a set of points, its MCC can either be determined by two points whose line segment connecting them makes up the diameter of a circle, or three points lie on the boundary of the circle. If it is determined by three points, then those three points must make up an acute triangle. This result indicates that there are at least two or three vertices lying on the boundary of the MCC of a GC. Fang Y, etc. [4] use this method to decide a spatial-aware community. We follow their approaches in this work.

Algorithm 2 GCExaminate takes vertices list $List \Lambda$ as input. It enumerates all the combinations of three vertices in the obtained $k$-core. After obtaining a circle defined by the three vertices (accomplished by function MCC), if its radius is less than $r$, then it examines nested $k$-core in the subgraph. If such a $k$-core exists, it will be inserted into list $ListKC$. Line 11 sort lists $ListKC$ by distance from the furthest to $q$ in ascending order in each $ListKC$.

In the following, we analyze the time complexity of our local search algorithm NGCSBasic and discuss the setting of an initial $d_1$ and appropriate $\alpha$.

From the definition of $k$-core [7], we know that a $k$-core community contains at least $k+1$ vertices. To retrieve $n$ such communities, the subgraph must contain at least $n + k$ vertices. Hence, the initial value of $d_1$ must be larger than $n + k$. Let $d^*$ be the optimal value of the subgraph. That is, vertex $ListV[d^*]$ is the last vertex needed for the subgraph containing $n$ GCs. Let $G_{d_h}$ be the subgraph examined in the $i$th iteration. Let $G_{d_0}$ be the subgraph NGCSBasic accesses before termination. We prove the time complexity of Algorithm 1 by the following lemmas.

**Lemma 1:** The time complexity of GCExaminate is $O(List \Lambda^3)$.

**Proof:** There are three nested for-loops in GCExaminate(). When it takes $List \Lambda$ as input, the time complexity of Algorithm 2 is $O(|List \Lambda|^3)$.  \[\square\]
Lemma 2: The time complexity of NGCSBasic is $O(\alpha^{-1}d^3_h)$.  
Proof: Algorithm 1 constructs and examines a series of subgraphs, $G_{d_1}, G_{d_2}, \ldots, G_{d_h}$. Each subgraph $G_{d_i}$ can be extracted from $G$ in $O(d_i)$ time. It needs $O(d_i)$ time to find a $k$-core in graph $G_{d_i}$. However, the time complexity of calling GCExamine is $O(d_i^3)$. By Lemma 1, we calculate the time complexity of NGCSBasic as follows:

$$O\left(\sum_{i=1}^{h} d_i^3\right) = O(d_1^3 + d_2^3 + d_3^3 + \ldots + d_h^3)$$

$$= O\left(\frac{d_1^3}{\alpha^{h-1}} + \frac{d_2^3}{\alpha^{h-2}} + \frac{d_3^3}{\alpha^{h-3}} + \ldots + \frac{d_h^3}{\alpha^1} + d_h^3\right)$$

$$= O\left(\frac{\alpha^2}{\alpha - 1}d_h^3\right) \quad (1)$$

Hence, Lemma 2 holds.

Theorem 1: The time complexity of NGCSBasic is $O(\frac{\alpha^2}{\alpha - 1}d^3)$.

Proof: It is easy to see that $d_{h-1} < d^* < d_h$. Moreover, we have $d_h \leq ad_{h-1} < ad^*$. Then, by Lemma 2, we have

$$O\left(\frac{\alpha}{\alpha - 1}d_1^3\right) = O\left(\frac{\alpha^2}{\alpha - 1}d^3\right) \quad (2)$$

This completes the proof.

Following Theorem 1, since we know that $\frac{\alpha^2}{\alpha - 1}$ has the smallest value when $\alpha = 2$ among all $\alpha$ larger than 1, we set $\alpha$ as 2 in this paper.

IV. A PROGRESSIVE LOCAL SEARCH APPROACH

The first limitation of NGCSBasic is its high computational cost, which makes it impractical for large spatial graphs. If we set $\alpha$ to 2 in Algorithm 1, the subgraph it searches will be more than four times larger than the optimal one according to the theorem 1. The second reason motivating us to devise a more efficient method is that Algorithm 1 only reports results at the end of its running. Thus, there is a long latency between issuing a query and obtaining any result.

In this section, we propose techniques to retrieve GCs progressively with increasing distance to the query vertex. It is also interesting to find the byproducts of such an approach, for the algorithm returns GCs one-by-one, such that a user can terminate it once having enough results.

We first define the notion of the keynode of a GC in the following:

Definition 5: (Keynode of a GC)

A vertex $u$ in a graph $G$ is a keynode of a GC regarding a query node if: (1) There exists a subgraph $G(V_u)$ of $G$ such that in $G(V_u)$ there exists a $k$-core and is covered by its MCC with a radius less than $r$. (2) Vertex $u$ is the furthest vertex in a GC to the query node.

For example, $v_4$ in Figure 1 is a keynode regarding query node $v_{10}$ when $k = 3$. Since the subgraph induced by vertices $\{v_4, v_5, v_8, v_9\}$ exhibits a 3-core and is covered by a circle with a radius less than $r$, $v_4$ is the furthest vertex to node $q$. In other words, there is a qualified GC in graph $G(V_u)$, and all the other vertices in the GC have a shorter distance to vertex $q$ than vertex $v_4$. In the same manner, we can observe that vertex $v_3$ is also a keynode regarding vertex $q$, which is from the subgraph induced by vertices $\{v_2, v_3, v_6, v_7\}$.

Because of nested $k$-cores in the graph, it is easy to see that there is no one-to-one correspondence between GCs and keynodes for a specific query node $q$. However, since we know that a keynode is the furthest vertex in a GC to the query vertex $q$, it is obvious that all the other vertices are located in an area between the keynode and vertex $q$. Thus, we have the following lemma, which helps us narrow the range where a GC can reside.

Lemma 3: Given a GC of query node $q$, and the GC’s keynode $u$ and subgraph $G(V_u)$ where the GC resides, we have $V_u \setminus u = \{v \in V, v \neq u & |q, v| < |q, u| & |u, v| \leq 2r\}$

Proof: For a vertex in $V_u \setminus u$, in equation $|q, v| \leq |q, u|$ directly follows from Definition 5. Vertex $v$ is also covered by a circle with a radius smaller than $r$ by Definition 3. This means that the longest distance from $v$ to $u$ must be $2r$. Thus, the lemma holds.

For example, $v_3$ in Figure 2 is the keynode of GC $\{v_2, v_3, v_6, v_7\}$. It is easy to see that vertices $v_2$, $v_6$ and $v_7$ are located in the gray overlay area, which is restricted by the formula $|v_{10}, v| < |v_{10}, v_3| & |v_3, v| \leq 2r$.

With Lemma 3, we now introduce our progressive local search approach. Recall that Algorithm 1 first invokes GCExamine to retrieve every possible GC in the area until there are no GCs found. Upon the termination of the algorithm, the subgraph it examines is usually larger than the optimal one, as convinced by Theorem 1. It is obvious that a larger subgraph will lead to more useless searching. Our key observation is that vertices other than keynodes in a GC are located in a restricted area. Therefore, we search for possible GCs only in this area, which will shorten the time needed to obtain the desired GCs. We produce the following algorithms for this purpose.

![Figure 2. Keynode $v_3$ and a GC belongs to.](image-url)
Algorithm 3 NGCSProgressive.

Input:
A graph $G$, sorted vertex list $ListV$, vertex $q$ and radius $r$

Output:
A $k$-core list $ListKC$

1: Initialize $ListKC$;
2: while ListV $\neq$ Null & L.length $< n$ do
3: \hspace{1em} $p = ListV.pop();$
4: \hspace{1em} $V' = \{v \in V & \exists q, v \leq |q, p| \& |p, v| \leq 2r\};$
5: \hspace{1em} if $|V'| < k$ then
6: \hspace{2em} Continue;
7: \hspace{1em} Copy $G'$ ($V'$, $E'$) from $G$;
8: \hspace{1em} Detect $k$-core in $G'$;
9: \hspace{1em} Create vertex list $List\Lambda$ for $k$-core
10: \hspace{1em} Sort list $List\Lambda$;
11: \hspace{1em} $ListKC.append(GCExamineP(List\Lambda, p, r));$
12: \hspace{1em} return $ListKC$;

The procedure for conducting GC examination also changes. Because keynode is fixed in Algorithm 4, the procedure only needs to check the combination of the two vertices in the list. Compared to Algorithm 2, the computational cost is reduced.

For example, in Figure 2, the sorted vertices list is $<v_6, v_1, v_2, v_9, v_7, v_5, v_8, v_3, v_4>$. The algorithm first examines vertices before $v_3$, that is, vertices $v_6, v_1, v_2, v_9, v_7, v_5, v_8$, and finds that there are not enough vertices to consist of a 3-core. In regard to $v_3$'s turn, there are $v_2, v_6$ and $v_7$. These three vertices, together with $v_3$, can make up a 3-core. Following $v_3$, vertex $v_4$ has $v_5, v_8$ and $v_9$. These four vertices can also make up a 3-core. Finally, Algorithm 3 returns these two GCs.

We show the time complexity of Algorithm 4 by Lemma 4.

Lemma 4: The time complexity of GCExamineP is $O(|List\Lambda|^2)$.

Proof: Algorithm GCExamineP takes $List\Lambda$ as input. It has two nested for-loops. Thus, the time complexity is $O(|List\Lambda|^2)$.

Lemma 4 shows that the time complexity of GCExamineP is one order less than that of GCExamine. This decrease will be illustrated in our experimental results in Section VI. Following Lemma 4, we have a time complexity for Algorithm 3.

Theorem 2: The time complexity of NGCSProgressive is $O(\frac{4}{3}d^3)$

Proof: Let the $d^{th}$ vertex in ListV be the last vertex examined in all iterations. From the previous discussion, it is easy to learn that the NGCSProgressive starts from the $(k+1)^{th}$ vertex in ListV. Therefore, the total running time for GCExamineP in NGCSProgressive is

$$O\left(\sum_{i=k+1}^d d_i^3\right) = O((k+1)^3 + (k+2)^3 + (k+3)^3 + \ldots + d^3)$$

$$= O\left(\frac{1}{3}d^3\right)$$

This completes the proof.

V. A FAST APPROXIMATE ALGORITHM

The NGCSBasic and NGCSProgressive algorithms return all the GCs related to a keynode. However, sometimes we only need to know one of these GCs for a specific keynode. For example, in social marketing applications, since vertices in a minimum covering GC are the spatially closest and the most likely to accept advertising, knowing such a GC is enough. NGCSProgressive is faster than the basic algorithm but it is still inefficient for large graphs since its time complexity is still cubic. In this section, we propose a fast approximate algorithm called NGCSApproximate.

Unlike NGCSBasic and NGCSProgressive, which examine every circle fixed by the vertices in the area, NGCSApproximate takes a strategy similar to SAC [4], which separates the search space into equal-sized cells and examines the circle centres in each cell. The logic behind this approach is to locate an approximate centre of a covering circle and examine the GC existing in the circle.

The approach is illustrated in Figure 3. The algorithm splits the search area into many $\gamma \times \gamma$ cells, as shown on the left side of Figure 3. We suppose that $G \{v_2, v_3, v_6, v_7\}$ is covered by a minimum circle $O(u, r_{opt})$. It is easy to see that locating an optimally covering circle takes time since the number of possible combinations of vertices to be explored can be infinite. However, by examining the finite cells in the overlap area, we can obtain an approximate circle located at the centre of a cell. For example, a circle $O(u, r_{opt} + \epsilon)$ in
is the centre of a cell and \( r \) half of \( \sqrt{\frac{2}{3}} \gamma \). This leads to an accelerated searching process and an approximate answer.

Because point \( u \) is the nearest centre point of cells for point \( u' \), we can see from Figure 3(b) that \( \varepsilon \leq \sqrt{\frac{2}{3}} \gamma \).

**Figure 3. A fast approximate algorithm.**

Algorithm 6 GCCell examines circle \( O(u, r + \varepsilon) \), where \( u \) is the centre of a cell and \( \varepsilon = \sqrt{\frac{2}{3}} \gamma \). If there is a \( k \)-core in the circle (Line 1), it will perform a binary search for a smaller circle. The algorithm takes the initial radius of the circle as the upper bound and equation 4 as the lower bound.

\[
l = \max\{|u, p|, \max_{v \in \text{KNN}(u)} |u, v|\},
\]

Algorithm NGCSApproximate takes the sorted vertex list \( \text{ListV} \), vertex \( q \) and radius \( r \) as input. When it does not exhaust \( \text{ListV} \) and has not obtained \( n \) minimum covered GCs, it executes the code in the while loop. It pops a vertex \( p \) in Line 3 and checks if there are enough vertices available to make up a \( k \)-core (Lines 3-6).

**Algorithm 5 NGCSApproximate.**

**Input:**
- A graph \( G \), sorted vertex list \( \text{ListV} \), vertex \( q \) and radius \( r \)

**Output:**
- A \( k \)-core list \( \text{ListKC} \)

1. Initialize \( \text{ListKC} \);
2. while \( \text{ListV} \neq \text{null} \) and \( \text{ListKC}.\text{length} < n \) do
3. \( p = \text{ListV}.\text{pop}() \);
4. \( V' = \{ v | v \in V \cup \{q, v\} < |q, p| \, \&\, |p, v| \leq 2r \} \);
5. if \( |V'| < k \) then
6. Continue;
7. Copy \( G' = (V', E') \) from \( G \);
8. Separate semicircle of \( O(p, r) \) faces \( q \) into cells;
9. Initialize \( \text{ListAS} = \) centre point of the cells;
10. Sort \( \text{ListAS} \) in random order;
11. Initialize \( \Lambda; \min = \infty \);
12. for \( i = 1 \) to \( t \) do
13. \( \Lambda', \min' = \text{GCCell}(\text{ListAS}[i], p, r) \);
14. if \( \Lambda' \neq \emptyset \) then
15. \( \Lambda = \Lambda' \);
16. \( \min = \min' \);
17. for \( i = t + 1 \) to \( \text{ListAS}.\text{length}() \) do
18. \( \Lambda', \min' = \text{GCCell}(\text{ListAS}[i], p, r) \);
19. if \( \Lambda' \neq \emptyset \) and \( \min' < \min \) then
20. \( \Lambda = \Lambda' \);
21. \( \min = \min' \);
22. Break;
23. if \( \min \neq \infty \) then
24. Insert \( \Lambda \) into \( \text{ListKC} \);
25. return \( \text{ListKC} \);

**Algorithm 6 GCCell.**

**Input:**
- Vertex \( u, p, \) radius \( r \)

**Output:**
- A \( k \)-core and radius of its covering circle

1. \( \Gamma = \) the \( k \)-core inside \( \text{Circle}(u, r + \varepsilon) \);
2. if \( \Gamma \neq \emptyset \) then
3. Initialize \( l = \max\{|u, p|, \max_{v \in \text{KNN}(u)} |u, v|\} \);
4. Initialize \( r_t = (r + l)/2 \);
5. while \( r - r_t > \beta \) do
6. \( V' = \{ v | v \in \text{Circle}(u, r_t) \, \&\, v \in \Gamma \} \);
7. \( \Gamma_t = \) the \( k \)-core containing \( u, p \) in \( G(V') \);
8. if \( \Gamma_t \neq \emptyset \) then
9. \( \Gamma = \Gamma_t \);
10. \( r = r_t \);
11. else
12. \( l = r_t \);
13. \( r_t = (r + l)/2 \);
14. Return \( \Gamma, r \)

Then, it splits the overlap section of circles \( O(q, |q, p|) \) and \( O(p, 2r) \) into cells, keeps the centre points of all these cells in a list (Line 9) and sorts them in random order (Line 10).
The algorithm first examines \( t \) cells and obtains the minimum coverage \( GC \) in these cells in Lines 12-16. It searches for the first \( k \)-core in the remaining cells, which is smaller than the recorded smallest covered \( k \)-core in the preceding \( t \) cells (Lines 17-22).

The selection of a positive integer \( t < |\text{ListAS}| \) is critical. Because the algorithm examines the first \( t \) cells, and takes the first cell thereafter that has a smaller \( k \)-core than all preceding cells. If it turns out that the minimum covered \( k \)-core is either among the first \( t \) cells (the worst case, with a possibility \( t/|\text{ListAS}| \)) or the \( i \)th cells \( (t < i < |\text{ListAS}|) \). We have the following theorem for Algorithm 5.

**Theorem 3:** In the \( j \)th iteration of Algorithm NGCSApproximate, if there is a qualified minimum covered GC, then it will be returned with a probability of at least \( 2/e \), and the radius of the covering circle is less than \( r_{opt} + \frac{\sqrt{2} \gamma + \beta}{2} \).

**Proof:** Let \( n \) be the length of \( \text{ListAS} \) and \( S \) be in the event that we succeed when the smallest-covered \( k \)-core is the \( i \)th cell examined. Note that the expected cell is either one of the first \( t \) cells or the \( i \)th cells \( (t < i < n) \), we have:

\[
Pr\{S\} = \sum_{i=1}^{n} Pr\{S_i\} = \sum_{i=1}^{t} Pr\{S_i\} + \sum_{i=t+1}^{n} Pr\{S_i\} = \frac{t}{n} + \sum_{i=t+1}^{n} Pr\{S_i\}
\]

(5)

When the expected cell is the \( i \)th, one or two things must happen. First, the smallest-covered \( k \)-core must be in position \( i \), and we denote this event by \( T_i \). Second, the algorithm must not select any of the cells in positions \( i \) through \( i-1 \); we denote the event that none of the cells in positions \( t+1 \) through \( i-1 \) are chosen by \( N_i \). Thus, we have

\[
Pr\{S_i\} = Pr\{T_i \cap N_i\} = Pr\{T_i\}Pr\{N_i\}
\]

(6)

The probability \( Pr\{T_i\} \) is clearly 1/\( n \). For event \( N_i \) to occur, the smallest-covered \( k \)-core in cells 1 through \( i-1 \) must be in one of the first \( t \) positions, and thus \( Pr\{N_i\} = t/(i-1) \). We have:

\[
\sum_{i=t+1}^{n} Pr\{S_i\} = \sum_{i=t+1}^{n} \frac{t}{n(i-1)} = \frac{t}{n} \sum_{i=t}^{n-1} \frac{1}{i}
\]

(7)

Approximating by integrals to bound equation 7 from above and below, evaluating these definite integrals, and differentiating with respect to \( t \), we have \( \frac{1}{n}(\ln n - \ln t - 1) \). When \( t = n/e \), the probability is maximized. Thus, we have:

\[
Pr\{S\} = \frac{t}{n} + \sum_{i=t+1}^{n} Pr\{S_i\} \geq \frac{t}{n} + \frac{1}{n}(\ln n - \ln t) = 2/e
\]

(8)

By Lemma 5, we complete the proof.

---

**TABLE 2. Statistics of social networks.**

| Dataset       | vertices | edges    |
|---------------|----------|----------|
| Facebook      | 4,039    | 88,234   |
| Brightkite    | 58,228   | 214,078  |
| Gowalla       | 196,591  | 950,327  |
| Youtube       | 1,134,890| 2,987,624|
| Wikipedia     | 2,394,385| 5,021,410|
| LiveJournal   | 3,997,962| 34,681,189|

**TABLE 3. Statistics of road networks.**

| Dataset                | vertices | Longitude | Latitude |
|------------------------|----------|-----------|----------|
| New York City          | 264,346  | [40.3; 41.3] | [73.5; 74.5] |
| Colorado               | 435,666  | [37.0; 41.0] | [102.0; 109.0] |
| Florida                | 1,070,376| [24.0; 31.0] | [79.875] |
| California and Nevada  | 1,890,815| [32.5; 42.0] | [114.0; 125.0] |
| Great Lakes            | 2,758,119| [41.0; 50.0] | [74.0; 93.0] |
| Western USA            | 6,262,104| [27.0; 30.0] | [100.0; 130.0] |

In NGCSApproximate, if we let the \( d \)th vertex in \( \text{ListV} \) be the last vertex that is examined in all iterations, the total running time for the algorithm is \( O(d^n \log \frac{n}{d}) \).

---

**VI. EXPERIMENT**

In this section, we evaluate the effectiveness and efficiency of our algorithms through extensive performance studies. The algorithms are compiled with Python 3.7. All the experiments are performed on a machine (Ubuntu Linux 64-bit 14.04 LTS) with an Intel i7 2.2 GHz CPU and 16 GB main memory. The results shown for each experiment are the averages of 100 independent tests.

---

**A. EXPERIMENTAL SETUP**

1) **DATASETS**

Six real social networks\(^1\) Facebook (FB), Brightkite (BR), Gowalla (GO), YouTube (YO), Wikipedia (WI), Livejournal (LI) and six real road networks\(^2\) California and Nevada (CAL), Colorado (COL), Florida (FLA), New York City (NY), Great Lakes (GL), Western USA (WU) are considered in our experiments. Table 2 and Table 3 show the characteristics of the datasets that we used in our experiments. In social networks, each vertex represents a user, and each edge represents the relationship between two users. For example, Facebook dataset consists of 'circles' (or 'friends lists') from Facebook social network, YouTube dataset consists of user-defined groups etc. In a road network, each vertex represents a spatial location, and each edge represents the distance between two locations.

Since locations in social networks are distributed globally around the world, we normalize the longitude and latitude of the locations of both networks to a two-dimensional \([0,1]\) space and then map users to the nearest intersection or

---

\(^1\)http://snap.stanford.edu/data/index.html

\(^2\)http://www.diag.uniroma1.it/challenge9/download.shtml
TABLE 4. Description of parameter.

| Parameter                          | Values                          |
|-----------------------------------|---------------------------------|
| The degree constraint of vertices $k$ | $3, 4, 5, 6, 7, 8, 9, 10$      |
| The community number $n$          | $3, 5, 7$                      |
| The spatial radius constraint $r$ | Dynamic values                 |
| The growing ratio of NGCS-B $\alpha$ | $1.5, 2, 3, 5, 10, 20$         |
| The length of the sides of cells $\gamma$ | $1, 8, 16, 32, 64, 128$       |

segment in the road network based on their coordinates. Moreover, we use FB+NY to represent the road-social network composed of Facebook and New York City, BR+COL to represent Brightkite and Colorado and GO+FLA to represent Gowalla and Florida.

2) ALGORITHMS

We conduct extensive performance studies to evaluate the effectiveness and efficiency of our local search framework and algorithms. Regarding the methods for NGCS, we evaluate the following algorithms.

- Global: Global finds the $k$-core community containing $q$ and uses the minimum degree metric for structure cohesiveness (Let $q$ be a query vertex). [19]
- NGCS-B: NGCSBasic method; A local search approach based on exponential growth strategy, we iteratively increase the size of the subgraph $G'$, with a growing ratio of $\alpha$ (Algorithm 1 and 2).
- NGCS-P: NGCSProgressive method; A progressive local search approach (Algorithm 3 and 4).
- NGCS-A: NGCSApproximate method; An approximation algorithm to improve the performance and efficiency (Algorithm 5 and 6).

3) PARAMETERS

We conducted experiments in different settings by varying six parameters, including degree constraint $k$, spatial radius constraint $r$, community number $n$, growing ratio of $\alpha$ and length of cell $\gamma$. In detail, Table 4 depicts the parameter settings in our experiments, where the bold numbers are default parameter values. In each set of experiments, we vary one parameter while setting the other parameters to default values.

B. EFFECTIVENESS EVALUATION

In this section, we first compare the NGCS algorithms with state-of-the-art methods and then study the average clustering coefficient and DistPr approximation of NGCS-A.

1) COMPARISON WITH THE STATE-OF-THE-ARTS

In this subsection, we show that the NGCS algorithms return communities with higher spatial cohesiveness than the state-of-the-art community retrieval methods: Global [19].

We introduce two metrics as follows:

- average clustering coefficient: overall level of clustering in a network.
- DistPr: average pairwise distance of vertices of the graph. Intuitively, higher values of the average clustering coefficient imply that it achieves higher social cohesiveness, and lower values of DistPr for a community imply that it achieves higher spatial cohesiveness. To compare the global method, we consider both NGCS-B and NGCS-A algorithms. We search communities using these algorithms and compute the average values of the above metrics for these communities. We report the results on six datasets in Figure 4. We can see that the average clustering coefficient of NGCS-A is slightly lower than those of the global and NGCS-B algorithms in Figure 4(a) since NGCS-A takes an approximate strategy and trades some approximation quality for efficiency. Additionally, because NGCS-B and NGCS-A take a location-based strategy, we can see that they have a smaller DistPr than the global algorithm in 4(b).

FIGURE 4. Comparison with existing method.

2) AVERAGE CLUSTERING COEFFICIENT AND DISTPR APPROXIMATION

In Figure 5, we report the average clustering coefficient and DistPr of NGCS-A for varying $\gamma$ on the Facebook, Brightkite, Gowalla and LiveJournal datasets. The average clustering coefficient of the communities is reported in Figure 5(a), where we vary the cell length $\gamma$. Note that if we set $\gamma = 1$, the results of NGCS-A are the same with those of NGCS-P. The average clustering coefficient of the detected communities obtained by our approximate method are very close to the optimal method. Figure 5(b) shows the DistPr of the detected communities. NGCS-A can detect the DistPr of communities that are very close to the optimal method by searching over a small graph in a progressive local search approach. Thus, NGCS-A balances efficiency and effectiveness well.

FIGURE 5. Varying $\gamma$ for NGCS-A.
C. EFFICIENCY EVALUATION

We investigate the efficiency of the proposed search algorithms and then compare each under different settings.

1) EFFICIENCY PERFORMANCE OF THE DEGREE CONSTRAINT $k$

First, we evaluate the performance of the solutions for NGCS when varying the degree constraint $k$ from 3 to 10. Figure 6 reports the running time of the three solutions on different datasets. The running time of these solutions increases clearly as the value of $k$ grows. The reason is that, for a larger value of $k$, the community cohesiveness is much larger, which results in a higher cost. Moreover, the efficiency of NGCS-P and NGCS-A becomes better for larger $k$. Additionally, we can see that NGCS-A performs the best among the 3 solutions and achieves a speedup of 5x to 6x over the baseline solution for all test cases. This is because it takes a strategy that separates the search space into equal-sized cells and examines the circle centres in each cell, which has a lower time complexity.

2) EFFICIENCY PERFORMANCE OF THE COMMUNITY NUMBER $n$

We then investigate the performance of the solutions by varying the community number $n$ from 3 to 7. We can see that the running time in Figure 7 ascends with an increasing $n$. Again, it is obvious that NGCS-A performs the best and is at least 4x faster than NGCS-B for all test cases. This is because NGCS-A conducts a $k$-core examination within MCC, which is made up of random cells near $O(d^* \log \frac{r}{\beta})$, while NGCS-B and NGCS-P need to compute MCC by enumeration, which is costly.

3) EFFICIENCY PERFORMANCE OF THE SPATIAL RADIUS CONSTRAINT $R$

Next, we evaluate the influence of the spatial radius constraint $r$. As illustrated in Figure 8, we can see that running time decreases with larger $r$ values. Our traversal algorithm is processed by for-loops in the examined algorithm (Algorithm 2, 4, 6). This procedure will loop more times when $r$ decreases to find the $n$ objective GC, which leads to an increase in running time due to the decrease in $r$. According to Lemma 3, the examining space of NGCS-P and NGCS-A is determined by $r$. Since the running time is almost the same when $r$ obtains a large value in the Facebook and Brightkite datasets, the improvement of NGCS-P and NGCS-A is more evident for a smaller $r$.

4) EFFICIENCY PERFORMANCE OF THE GROWING RATIO $\alpha$

Figure 9 reports the running time of NGCS-B with the growing ratio $\alpha$ on different datasets. If we could not find
enough results in the current area, we need to exponentially increase the size of the vertex by increasing the ratio $\alpha$. As illustrated in Fig. 9, the running time generally increases as $\alpha$ increases. The reason is that a larger $\alpha$ means we need to search for more communities satisfying the above three criteria. Recall from Section 3 that according to our analysis in time complexity, the NGCS-Basis performs best for $\alpha$ when it is approximately 2, as shown in Figure 9.

5) EFFICIENCY PERFORMANCE OF THE CELL LENGTH $\gamma$

Recall from Section 5 that NGCS-A separates the search space into equal-sized cells and examines the circle centres in each cell; we set an approximate point as the centre of a covering circle. As a result, NGCS-A trade accuracy in return for increased performance. Figure 10 illustrates the NGCS-A performance when we vary the value of $\gamma$ to 8, 16, 32, 64 and 128. We can see that the running time decreases with large $\gamma$ values. Nevertheless, the larger $\gamma$ is, the more the approximation quality is traded for efficiency. We need to set proper $\gamma$ to balance efficiency and effectiveness when solving specific problems.

VII. CONCLUSION

In this paper, for the first time, we have studied the problems of Neighboring Geosocial Communities Search over Large Spatial Graphs, which searches communities of users that are socially and spatially close to each other. To efficiently address this problem, we proposed a baseline solution and two effective progressive methods. Extensive empirical studies on large-scale real-world location-based social networks demonstrate that our proposed methods substantially outperform the baseline methods based on an exponential growth strategy under various system settings.

REFERENCES

[1] X. Huang, L. V. S. Lakshmanan, and J. Xu, “Community search over big graphs,” Synth. Lectures Data Manage., vol. 14, no. 6, pp. 1–206, Aug. 2019.
[2] X. Huang, L. V. S. Lakshmanan, J. X. Yu, and H. Cheng, “Approximate closest community search in networks,” Proc. VLDB Endowment, vol. 9, no. 4, pp. 276–287, 2015.
[3] G. Palla, I. Derényi, I. Farkas, and T. Vicsek, “Uncovering the overlapping community structure of complex networks in nature and society,” Nature, vol. 435, p. 814, Jun. 2005.
[4] Y. Fang, R. Cheng, X. Li, S. Luo, and J. Hu, “Effective community search over large spatial graphs,” Proc. VLDB Endowment, vol. 10, no. 6, pp. 709–720, 2017.
[5] L. Chen, C. Liu, R. Zhou, J. Li, X. Yang, and B. Wang, “Maximum co-located community search in large scale social networks,” Proc. VLDB Endowment, vol. 11, no. 10, pp. 1233–1246, Jun. 2018.
[6] J. Kim, T. Guo, K. Feng, G. Cong, A. Khan, and F. M. Choudhury, “Densely connected user community and location cluster search in location-based social networks,” in Proc. ACM SIGMOD Int. Conf. Manage. Data, Jun. 2020, pp. 2199–2209.
[7] S. B. Seidman, “Network structure and minimum degree,” Social Netw., vol. 5, no. 3, pp. 269–287, 1983.
[8] M. Wang, C. Wang, J. X. Yu, and J. Zhang, “Community detection in social networks: An in-depth benchmarking study with a procedure-oriented framework,” Proc. VLDB Endowment, vol. 8, no. 10, pp. 998–1009, 2015.
[9] W. Cui, Y. Xiao, H. Wang, Y. Lu, and W. Wang, “Online search of overlapping communities,” in Proc. ACM SIGMOD Int. Conf. Manage. Data, 2013, pp. 277–288.
[10] P. Lee and L. V. Lakshmanan, “Query-driven maximum quasi-clique search,” in Proc. SIAM Int. Conf. Data Mining, 2016, pp. 522–530.
[11] X. Huang, L. V. S. Lakshmanan, J. X. Yu, and H. Cheng, “Approximate closest community search in networks,” 2015, arXiv:1505.05956.
