Densely Connected User Community and Location Cluster Search in Location-Based Social Networks

Junghoon Kim
Nanyang Technological University, Singapore
junghoon001@e.ntu.edu.sg

Tao Guo
Google, Singapore
darkgt@google.com

Kaiyu Feng
Nanyang Technological University, Singapore
kfeng002@e.ntu.edu.sg

Gao Cong
Nanyang Technological University, Singapore
gaocong@ntu.edu.sg

Arijit Khan
Nanyang Technological University, Singapore
arijit.khan@ntu.edu.sg

Farhana M. Choudhury
University of Melbourne, Australia
farhana.choudhury@unimelb.edu.au

Abstract

Searching for a community based on query nodes in a graph is a fundamental problem and has been extensively investigated. Most of the existing approaches focus on finding a community in a social network, and very few studies consider location-based social networks where users can check in locations. In this paper we propose the GeoSocial Community Search problem (GCS) which aims to find a social community and a cluster of spatial locations that are densely connected in a location-based social network simultaneously. The GCS can be useful for marketing and user/location recommendation. To the best of our knowledge, this is the first work to find a social community and a cluster of spatial locations that are densely connected from location-based social networks. We prove that the problem is NP-hard, and is not in APX, unless $P = NP$. To solve this problem, we propose three algorithms: core-based basic algorithm, top-down greedy removing algorithm, and an expansion algorithm. Finally, we report extensive experimental studies that offer insights into the efficiency and effectiveness of the proposed solutions.

CCS Concepts

• **Information systems → Clustering**.

Keywords

community search; location-based social network analysis

SIGMOD'20, June 14–19, 2020, Portland, OR, USA

ACM ISBN 978-1-4503-6735-6/20/06…$15.00
https://doi.org/10.1145/3318464.3380603

ACM Reference Format:
Junghoon Kim, Tao Guo, Kaiyu Feng, Gao Cong, Arijit Khan, and Farhana M. Choudhury. 2020. Densely Connected User Community and Location Cluster Search in Location-Based Social Networks. In *Proceedings of the 2020 ACM SIGMOD Int'l Conference on Management of Data (SIGMOD '20), June 14–19, 2020, Portland, OR, USA.* ACM, Portland, OR, USA, 11 pages. https:/doi.org/10.1145/3318464.3380603

1 Introduction

Informally, a community is a subgraph where the nodes are densely connected internally, and sparsely connected externally. Community detection problem is to identify all the communities in a graph.

In contrast, given a set of query nodes and some other constraints, community search problem aims to find a densely connected community containing all query nodes [3, 11, 12, 17, 19, 28]. It is different from community detection since the query nodes are given and the goal is to find one community.

Many location-based social network (LBSN) services have emerged in recent years. For example, Foursquare hosts more than 12 billion check-ins and 105 millions venues[1]. LBSN contains social network and check-in information. One characteristic of LBSN is Preference locality [22]. It means that individuals prefer nearby locations for different activity purposes [15]. Figure 1 depicts an example of an LBSN.

In this work, we focus on finding a community and a cluster in the LBSN, consisting of a group of users and a set of locations, respectively, that are densely connected. We aim to find the community and cluster with high check-in density, since the frequent visiting behaviour implies the strong intention of common interests for a specific group of people. Meanwhile, the users in the result community should also be firmly connected socially and their visiting locations should be close spatially. To this end, we model the closeness with structural/spatial constraints. Specifically, we

[1]https://foursquare.com/about

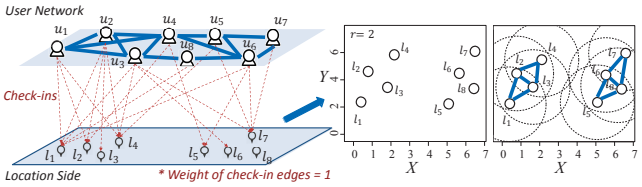

Figure 1: Location-based social network and location graph generation

propose the GeoSocial Community Search problem (GCS) which, given a location-based social network and a set of query nodes, aims to find a subgraph containing the query nodes such that the users in the subgraph are socially close and they frequently check-in to a cluster of closeby locations.

Applications. GCS has a wide range of applications.

(1) User Recommendation [13] Online review services like Yelp provide a platform to explore and find restaurants, bars and more. Such services also enable business owners to target ads to their potential customers. Suppose that the owner of a restaurant wants to attract diners by giving away coupons to potential customers. GCS can be used to find the community of users who have frequently visited nearby locations of the restaurant. They will be the ideal target customers, since they are more likely to visit the restaurant as a group of friends. Furthermore, the restaurant can form partners with those nearby locations frequently visited by the user community for cross-promotion.

(2) Event/Location Recommendation and Organization [24, 26] GeoSocial applications such as Meetup, Meetin, and Eventbrite support users to organize events in a physical location, and recommend events to users. First, to recommend events to Alice and Bob, GCS can be used as follows: with Alice and Bob, as well as their current locations, as query nodes, GCS is used to find those locations that are close to their current locations, and have been visited frequently by a community of users containing both Alice and Bob. Then we can recommend Alice and Bob the events that will happen in those locations and are organized by users in the community, since very likely they are of interest to Alice and Bob. Second, to help Alice and Bob organize an event, GCS can be employed to find the community containing Alice and Bob, as well as the set of locations frequently visited by the community. Thus, candidate invitees can be from the community, the community size can be used as an estimator of the participants for event planning, and the event location can be selected from the set of locations.

(3) GeoSocial Data Analysis Studying features of a social network or geographical regions is an interesting problem in data analysis. With GCS, we can infer characteristics of a community (resp. region) from its region (resp. community). For instance, if the region contains sport facilities such as gym or swimming pool, we expect that the users in the corresponding community are interested in sports. Similarly, by analyzing members of a community (e.g., comprising many IT people), it is possible to better understand the characteristics of a geographical area (e.g., high-tech region).

The contributions of our work are summarized as follows:

(1) **Define the community search problem in LBSNs.** To the best of our knowledge, this is the first work to find a community of social network users and a cluster of locations with high check-in density. We have also suggested applications of this problem.
(2) **Theoretical Analysis.** We show that the GCS problem is NP-hard, and is not in APX.
(3) **Solutions.** We propose three solutions to solve the GCS problem effectively and efficiently.
(4) **Extensive Evaluation.** We conduct extensive experiments on real-life networks in terms of efficiency and scalability. We also show the usefulness of the GCS problem with case study results.

2 Related Work

Community search problem is first introduced by Sozio et al. [28]. They propose a global search algorithm to find a subgraph that contains query nodes and show that community search with upper-bound constraints on the size of result is NP-hard. Cui et al. [10] analyse the challenge of global search that have to explore the whole graph and propose a local search algorithm starting from a query node. However, the algorithm can support only one query node. Barbieri et al. [3] propose a parameter-free community search algorithm. It aims to overcome the limitation of local search algorithms by allowing multiple query nodes. They use k-core decomposition and heuristic steiner tree algorithm to find a community. These proposals define community by minimum degree. In contrast, Huang et al.[17, 19] find k-truss community, where every edge in the community is contained in at least $(k-2)$ triangles. The algorithms proposed in these studies on community search cannot be used for our problem, which is different from these existing community search problems.

One extension of community search is to consider other information. For example, Fang et al. [12] consider structure cohesiveness and keyword cohesiveness. They claim that considering keywords helps better understanding of the formation of a community. Huang et al. [18] study the attribute-driven k-truss community search problem. They consider the pairwise distances of the nodes and aim to maximize attribute score. Spatial information is also considered as attributes of users in finding a community [7, 11, 29]. Fang et al. [11] assume that each user is associated with a single location, and identify a spatial-aware community (SAC). Given a query node, SAC is to find a k-core community in a minimum covering circle with the smallest radius, and the resultant

Table 1: Comparing GCS and related work

	GCS	SAC[11]	MCC[7]	(k,r)-core[30]
Check-in graph	Yes	No	No	No
Query type	users, locations, or both	a user	No	No
Result	a community and a cluster	a community	top-k communities	maximal (k,r)-core communities
Objective	maximize check-in density	minimize radius	maximize size	enum. all maximal (k,r)-core

community contains the given query node. Chen et al. [7] make the same assumption that each user is associated with a single location, and define a maximum co-located community (MCC) search problem. This problem is to find the largest k-truss community in which the pairwise distance is smaller than a threshold specified by the user. Zhang et al. [30] introduce (k,r)-core problem in an attributed social network, where each user has an attribute. They aim to enumerate all the maximal (k,r)-core, where each (k,r)-core comprises a set of densely connected users, which form a k-core community, and the similarity between the attributes of any two users exceeds a given threshold r; A (k,r)-core is maximal if none of its supergraphs is a (k,r)-core.

Our work is fundamentally different from SAC[11], MCC[7], and (k,r)-core [30] in at least four aspects as summarized in Table 1. First, our input is a check-in graph with two sides: social network and location, and each user can check-in multiple locations. Second, GCS can take both locations and users as query nodes. Third, our result comprises a community of users, a cluster of locations, and the check-in connections between them. Finally, our objective is to maximize the check-in density between the two levels of graphs. Therefore, SAC, MCC, and (k,r)-core cannot be used for the applications of GCS given in Introduction.

3 Problem and Basic Solution

3.1 Problem statement

A *social network* $G = (V, E)$ is a graph consisting of a set V of nodes and a set $E \subseteq V \times V$ of edges. In this study, we assume that all graphs considered in this work are simple and undirected. Given a subset of nodes $H \subseteq V$, we denote $G[H] = (H, E[H])$ the subgraph of G induced by H, i.e., $E[H] = \{(u,v) \in E | u, v \in H\}$. A bipartite graph $G_C = (V_U, V_L, E_C)$ is a *check-in graph*, where V_U is a set of user nodes, V_L is a set of locations, $(u,l) \in E_c$ with $W(u,v) \in \mathbb{R}$ being the weight of the edge $(u,v) \in E_C$. The check-in weight W can consider the frequency of check-in as well as the recency of the check-in. Finally, A *location-based social network (LBSN)* $S = (G_U, V_L, G_C)$ consists of a social network G_U, location data V_L, and check-in graph G_C.

DEFINITION 1. *[User Community]. A user community is a connected subgraph $H = (V_H, E_H) \subseteq G_u$ that satisfies $\delta(H) \geq m$, where $\delta(H)$ is the minimum degree of any node $v \in V_H$ and m is a user-specified degree threshold.*

We employ the minimum degree within a subgraph as its goodness measure, since it has been widely used in community search problems [3, 10–12, 28], and is generally utilized for measuring the structural cohesiveness [27].

DEFINITION 2. *[Distance Reachable]. Two locations l_1 and l_k are distance reachable if there exists a sequence of locations $< l_1, l_2, \cdots, l_k >$ such that $dist(l_i, l_{i+1}) \leq r$ for any $i \in [1, k-1]$.*

DEFINITION 3. *[Location Cluster]. Give a user-defined distance threshold r and degree threshold m, a set of location L form a cluster when the following constraints hold.*
(1) $\forall l \in L$, there are at least m nearby locations within r distance in L ;
(2) Any two locations in L are distance reachable.

DEFINITION 4. *[Location Network]. Given location data V_L and a user-defined distance threshold r, the location network $G_L = (V_L, E_L)$ is constructed as follows. When any two locations $l_i, l_j \in V_L$, $l_i \neq l_j$, are located within radius r (i.e., the distance between l_i and l_j, denoted as: $dist(l_i, l_j) \leq r$), we connect these two locations by an edge.*

Figure 1 demonstrates how to form a location graph with radius $r = 2$. Here, we choose a radius r to construct the network, which is intuitive — the closer the locations are, the more accessible they are from each other.

Note that in our algorithm, we do not materialize a complete location network. The concept of location network is for presentation convenience, and to see the similarity of the constraints used in user community and location cluster. With the concept of location network, we can see that the constraints for defining a location cluster are actually the minimum degree constraints on a connected location subgraph, which are the same as those for defining a community.

Table 2: Notations

Notation	Definition
community: G_{CU}	subgraph $G_{CU} \subseteq G_U$ w/ cohesiveness
cluster: G_{CL}	subgraph $G_{CL} \subseteq G_L$ w/ cohesiveness
$\delta(G)$	minimum degree in graph G
$\rho(G)$	check-in density of graph G
$E_C[S,H]$	check-in edge from H to S
$W_H[C]$	sum of check-in weight from C to H
$G[H]; H \subseteq V$	subgraph $G[H]$ of $G = (V,E)$, induced by node set H

For convenience, we call the constraints of minimum degree threshold and connectivity as *Cohesiveness constraint*. We say that a user community or a location cluster satisfies the cohesiveness constraint.

DEFINITION 5. *[Check-In Density]. Consider a check-in graph $G_C = (V_U, V_L, E_C)$. Given a user community $H_{CU} = (V_{CU}, E_{CU})$ and a location cluster H_{CL}, the check-in density $\rho(H_{CU}, H_{CL})$ between H_{CU} and H_{CL} is defined as $\rho(H_{CU}, H_{CL}) = \sum_{u \in H_{CU}, v \in H_{CL}} W(u, v)/(|V_{CU}| + |H_{CL}|)$.*

In Figure 1, let us consider a community induced by the users $V_{CU} = \{u_1, u_2, u_3\}$, and a cluster induced by the locations $V_{CL} = \{l_1, l_2, l_3, l_4\}$. The check-in density of the bipartite subgraph induced by $V_{CU} \cup V_{CL}$ is: 9/7.

We are now ready to define our problem. The notations are summarized in Table 2.

PROBLEM DEFINITION 1. *[GeoSocial Community Search (GCS)]. Given a location-based social network $S = (G_U, V_L, G_C)$, query nodes $Q \subseteq V_U \cup V_L$, a degree threshold m, and a radius r, the GCS problem aims at finding a community V_{CU} and a cluster V_{CL} containing all the query nodes Q such that the check-in density between V_{CU} and V_{CL} is maximized.*

Note that both m and r are input parameters, known only at query time. The intuition of our problem is as follows. Given a set Q of query nodes, we search for a user community V_{CU} and a location cluster V_{CL}, covering all query nodes and satisfying the cohesiveness constraints, such that the check-in density between the user community and the location cluster is maximized.

The GCS problem is designed for the applications given in Section 1. Both the cohesiveness constraints and the check-in density maximization are important for those applications.

3.2 Theoretical characterization

Our problem is, however, nontrivial. The following theorem shows that the GCS problem is intractable. For simplicity, we assume that the weight of all check-in edges to be one.

THEOREM 1. *GCS problem is* **NP***-hard.*

Proof. We prove that GCS problem is **NP**-hard by reducing an instance of the mCST(minimum Community Search with Threshold constraint) problem which is **NP**-hard to an instance of GCS in polynomial time. Given a graph $G = (V, E)$, a query node $q \in V$, and degree threshold m, the goal of mCST[10] is to find nodes $H \subseteq V$, such that (1) H contains q; (2) the subgraph $G[H]$ induced by H is connected; (3) the minimum degree in $G[H]$ is greater than or equal to m (i.e., $\delta(G[H]) \geq m$); and (4) the size of H is minimized.

Consider an instance of mCST:$G_1 = (V_1, E_1), m, q$, with $q \in V_1$. We construct a complete graph $G_2 = (V_2, E_2)$ having $|V_2| = m + 1$ nodes. We use G_1 as the social graph and G_2 as the location graph in the corresponding instance of the GCS problem . Next, we add a check-in edge between every node in V_2 and $q \in V_1$. Thus, we construct an instance of GCS problem : we select query nodes $Q = \{q\} \cup V_2$, and employ the same node degree threshold m. The optimal solution of GCS will have all nodes in V_2 as a cluster (due to cohesiveness constraints). Furthermore, assume that the community selected from G_1 in the optimal solution of GCS is induced by the subset of nodes $C_1 \subseteq V_1$. Then, this optimal solution has check-in density: $\frac{m+1}{|C_1|+(m+1)}$. Now, the solution has the maximum check-in density when $|C_1|$ is minimized. Recall that the subgraph induced by C_1 is a community, thus it satisfies the cohesiveness constraints. However, this is same as the mCST problem over G_1. Therefore, GCS problem is **NP**-hard due to the polynomial-time reduction from mCST. □

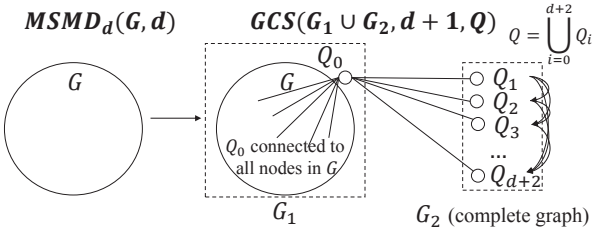

Figure 2: Construct an instance for GCS from MSMD$_d$.

THEOREM 2. *GCS is not in APX for any $m \geq 4$, unless $P = NP$.*

Proof. We give an L-reduction [9] from MSMD$_d$(Minimum Subgraph of Minimum Degree) [1, 2] to an instance of GCS problem in polynomial time. MSMD$_d$ problem is a minimization problem and is not in APX, when $d \geq 3$ and unless P=NP [1]. For convenience, we consider the inverse problem of GCS to make the minimization problem, which is because L-reduction preserves PTAS and APX if the original problem is the minimization problem[9]. Since the original problem MSMD$_d$ is not in APX when $d \geq 3$ and unless $P = NP$, our claim is that GCS problem is also not in APX, unless $P = NP$.

We show how to make an instance of GCS from the instance of MSMD$_d$ in Figure 2. The reduction step is as follows.

(1) It is known that MSMD$_d$ is not in APX [1, 2]. The problem is as follows : given a graph $G = (V, E)$ and a degree threshold d, the goal of MSMD$_d$ is to find a subgraph $H \subseteq V$, such that (1) the degrees of every node in H are larger than or equal to d, i.e., $\delta(H) \geq d$ (2) the size of the subgraph H is minimized. We note that the result of MSMD$_d$ is a connected component due to minimizing the size. Furthermore, the optimal solution of the MSMD$_d$ problem must have at least $d + 1$ nodes.
(2) Consider an instance of MSMD$_d$: $(G = (V, E), d)$. We construct a complete graph $G_2 = (V_2, E_2)$ with $V_2 =$

$\{Q_1, Q_2, \cdots, Q_{d+2}\}$ nodes and another graph $G_1 = (V_1, E_1)$ where $V_1 = V \cup Q_0$. Let the node $Q_0 \in V_1$ be connected to all the nodes in $V_1 \setminus Q_0$. We consider G_1 as the social graph and G_2 as the location graph in the corresponding instance of the GCS problem. Next, we add check-in edges between every node in V_2 and $Q_0 \in V_1$. The sum of check-in edge weights is exactly $d+2$. Thus, we construct an instance of the GCS problem (LBSN $S = G_1 \cup G_2$, query nodes $Q = \bigcup_{i=0}^{d+2} Q_i$ and the degree threshold $m = d + 1$).

(3) Based on generating the instance of GCS problem from MSMD_d, we know that the solution of GCS problem contains all nodes in V_2 and $Q_0 \in G_1$. This is because G_2 is a complete graph and $|V_2|$ is the minimum size to become a cluster. Our goal is to maximize the check-in density. To maximize the check-in density, we should find a community having a minimum size, i.e., the optimal solution has check-in density: $\frac{m+1}{|C_1|+(m+1)}$ where C_1 is a community in G_1. Now, it is easy to verify that the nodes in $C_1 \setminus Q_0$ correspond to the MSMD_d.

Now, we are ready to show L-reduction from MSMD_d to GCS. We switch GCS for convenience from maximization to minimization. New problem $\overline{\mathrm{GCS}}$ is to minimize the inverse check-in density. In this proof, we use following notations.

- x is an instance of MSMD_d.
- $f(x)$ is an instance of $\overline{\mathrm{GCS}}$.
- if y' is a solution to $f(x)$, then $g(y')$ is a solution to x.
- $OPT(.)$ defines the cost of the optimal solution for the given instance of a problem.
- $Sol'_{\mathsf{MSMD}_d}(g(y'))$ is the cost of the solution $g(y')$ for the instance x of MSMD_d.
- $Sol'_{\overline{\mathrm{GCS}}}(y')$ is the cost of the solution y' for the instance $f(x)$ of $\overline{\mathrm{GCS}}$.

There is an L-reduction (f, g) if two positive constants α and β exist such that

$$OPT_{\overline{\mathrm{GCS}}}(f(x)) \leq \alpha OPT_{\mathsf{MSMD}_d}(x) \quad (1)$$

$$|OPT_{\mathsf{MSMD}_d}(x) - Sol'_{\mathsf{MSMD}_d}(g(y'))| \leq \beta(|OPT_{\overline{\mathrm{GCS}}}(f(x)) - Sol'_{\overline{\mathrm{GCS}}}(y')|) \quad (2)$$

We find that there exist such $\alpha = 1$ and $\beta = (d+2)^2$. Equation 1 holds when we use $\alpha = 1$. Recall that $d \geq 3$ and $OPT_{\mathsf{MSMD}_d}(x)$ is always larger than or equal to $d + 1$.

$$\frac{OPT_{\mathsf{MSMD}_d}(x) + d + 3}{d+2} \leq OPT_{\mathsf{MSMD}_d}(x) \quad (3)$$

Equation 2 holds when we use $\beta = (d+2)^2$. Recall that $Sol'_{\overline{\mathrm{GCS}}}(y')$ is equal to the $\frac{Sol'_{\mathsf{MSMD}_d}(g(y'))+d+3}{d+2}$.

$$OPT_{\mathsf{MSMD}_d}(x) - Sol'_{\mathsf{MSMD}_d}(g(y')) \leq \beta(\frac{OPT_{\mathsf{MSMD}_d}(x) + d + 3}{d+2} - \frac{Sol'_{\mathsf{MSMD}_d}(g(y')) + d + 3}{d+2}) \quad (4)$$
$$= (d+2)(OPT_{\mathsf{MSMD}_d}(x) - Sol'_{\mathsf{MSMD}_d}(g(y'))$$

L-reduction preserves membership in APX and PTAS for the minimizing optimization case [9]. Since MSMD_d is not in APX for $d \geq 3$ and unless $P = NP$, our GCS problem is also not in APX for $m \geq 4$ and unless $P = NP$. □

Corollary 2.1. *The GCS problem is not in PTAS for $m \geq 4$, unless P=NP.*

Since PTAS is a subclass of APX and we have already shown that GCS is not in APX, the above corollary holds.

3.3 Basic Algorithm (BA)

Our basic method follows the notion of *core decomposition* [27] of a graph: The *k-core* of an undirected graph $G = (V, E)$ is a maximal subgraph $G[C_k] = (C_k, E[C_k])$ such that the minimum degree within the subgraph $\delta(G[C_k]) \geq k$. In other words, the k-core of a graph is a maximal subgraph in which every node is connected to at least k other nodes within that subgraph. The k-core has the following two properties. *Uniqueness*: The k-core of a graph G is unique. *Containment*: The $(k+1)$-core of a graph G is a subgraph of its k-core. The maximal k for which a node v belongs to the k-core is the *core index* of v. The set of all k-cores $V = C_0 \supseteq C_1 \supseteq \ldots \supseteq C_{k^*}$ ($k^* = \mathrm{argmax}_k\, C_k \neq \phi$) is the core decomposition of G. Based on the containment property, the $(k+1)$-core can be obtained by *peeling* the k-core. It means to recursively delete, from the k-core of G, all nodes with degree less than $k + 1$; what remains is the $(k+1)$-core of G [5].

While the k-core $G[C_k]$ of G, for a given k, is unique, $G[C_k]$ may not be a connected graph. To guarantee the connectivity, we define a set Γ of nodes as a *Core-Based Connected Component* iff the subgraph induced by Γ is a connected component and its minimum degree is larger than or equal to k. We refer to k as its core value and use $CBCC(k)$ to denote the set of $CBCC$s whose core values are k.

We notice that *every k-core-based connected component $\Gamma \in CBCC(k)$ satisfies the cohesiveness constraints if the input degree threshold $m \leq k$*. Our basic approach exploits this property. In particular, for a given degree threshold m, we compute all core-based connected components in $CBCC(k)$, for all $k \geq m$, both from the social network and the location network. Next, we discard those connected components from the user (resp. location) side, which do not contain all query user (resp. location) nodes. If there is no connected component, our algorithm returns null. Among the remaining connected components, we consider every pair (one from the user side and the other from the location side), compute

the check-in subgraph density for this pair, and finally report the pair having the maximum check-in subgraph density as the solution to our problem.

Generating the location graph. To generate the location graph, we use the R-tree [14]. Based on the query location, we incrementally extract a small portion of the location graph. If the query does not contain a location, we find all the locations visited by the subgraph which is the result of k-core containing all user query nodes. We then iteratively generate a location graph by including the locations as a set of seed nodes of the location graph.

Time complexity. The complexity of the BA is $O(|E_U| + |E_L| + m'_{G_U}(|V_U| + |E_U|) + m'_{G_L}(|V_L| + |E_L|) + |V_L|^2)$ where $|E_U|$ and $|E_L|$ are for finding core index for both networks [4]; $D(V + E)$ is for finding all $CBCC(k)$ in a network where m'_G is maximal core index in a network G; $|V_L|^2$ is for generating a location graph, which is usually much smaller in practice. Thus, the time complexity of the algorithm is $T_{BA} = O(m'_U(|V_U| + |E_U|) + m'_L(|V_L| + |E_L|) + |V_L|^2)$.

Difficulties with the basic algorithm. Even if the BA is efficient, it may return a very large community since the result is directly depending on the k-core. It is known that a community search problem that maximizes the minimum degree also returns a large community[10, 28]. Thus, our BA also suffers from very large solutions.

Leveraging the result of basic algorithm. In the following improved algorithms (GRA and GEA$^+$), we can leverage the result of the BA as an input graph instead of the original graph. Thus, Thus, we can reduce the running time (demonstrated empirically in Section 6) and get a better start point.

4 Greedy Removing Algorithm (GRA)

In this section, we propose a more effective top-down heuristic algorithm for solving the GCS problem.

Our algorithm is inspired by the top-down approach [6, 21] which finds the densest subgraph in a graph. However, these algorithms cannot be directly applied since in our problem we need to deal with query nodes Q, two layers of network, and the cohesiveness constraints imposed on the solution. The high-level idea of our algorithm GRA for finding the densest subgraph is to iteratively remove nodes which have the smallest average check-in degree by maintaining cohesiveness constraints. For every iteration, we compute the check-in density. Finally, we pick a subgraph that has the largest density. In each iteration, our algorithm iteratively removes a group of nodes instead of a single node. This is because after removing a node with the smallest check-in degree, the remaining graph should always satisfy the cohesiveness constraints. In order to satisfy the constraints for the remaining graph, it is necessary to delete the nodes

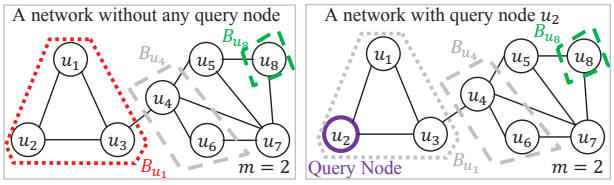

Figure 3: Toy networks with blocks B_{u_1}, B_{u_4}, and B_{u_8}

that do not satisfy the constraints together. The concept of "block" is inspired by this idea.

DEFINITION 6. *[DVS(Degree Violation Set)]. Given a network G and a node $w \in G$, we remove the node w from the network G. Then, to hold the degree constraint in the remaining graph $G \setminus \{w\}$, we may need to remove some other nodes. This process is iteratively repeated. We denote the set of removed nodes to hold the degree constraint by DVS(w).*

DEFINITION 7. *[Block]. Given a network $G = (V, E)$ and node $u \in V$, block B_u is the union of node u and DVS(u).*

To maintain the cohesiveness constraints and existence of all query nodes Q in the result, we next introduce the removable block.

DEFINITION 8. *[Removable Block]. Given a network $G = (V, E)$, query nodes $Q \subseteq V$, and a block $B_u \subseteq V$, the block B_u is removable if the block satisfies both conditions: (1) B_u does not contain any query node in Q (2) removing B_u from G will not induce multiple connected components.*

We briefly show how to generate blocks B_{u_1}, B_{u_4} and B_{u_8} from the toy network in Figure 3. We set the minimum degree threshold m as 2.

First, we check blocks in the network without any query node (left figure in Figure 3). Block B_{u_1} contains nodes $\{u_1, u_2, u_3\}$ since after removing node u_1, node u_2 does not satisfy the degree constraint and it needs to be deleted; After removing node u_2, we delete node u_3 in a cascading manner due to the degree constraint. Block B_{u_1} is removable since it satisfies both conditions in Definition 8. Block B_{u_4} contains $\{u_4, u_6\}$ and B_{u_4} is not a removable block since the result of removing block B_{u_4} has two connected components. Block B_{u_8} contains $\{u_8\}$ and is removable. Next, consider the same network with a query node u_2 (right figure in Figure 3). Block B_{u_1} is not a removable block since it contains query node u_2.

In our algorithm, we remove a block instead of a node to meet the cohesiveness constraints. We next define average check-in density for each block, which will be used to prioritize the blocks to be removed.

DEFINITION 9. *[Block check-in density]. Given a remaining check-in graph $H = (V_H, E_H)$ and a removable block $B \subseteq V_H$, the block check-in density $\rho(B, H)$ is $\frac{W_H[B]}{|V(B)|}$.*

Block check-in density is a measure about average check-in density from a block to the remaining graph H. Since the size of the graph H is reduced after each iteration, the check-in density of the blocks may be changed in each iteration.

GRA works as follows. Let H denote the result subgraph. First, we set the initial solution H satisfying the cohesiveness constraints from the LBSN. Then, we start removing process in H. We iteratively remove a block which is removable and has minimum block check-in density from H. In every iteration, the selected block will be removed from its network G_U or G_L and the currently remaining check-in network H'. Then, we compare the check-in density between H and H' to keep the densest subgraph and update H with H' if H' has a higher check-in density, i.e.,$\rho(H') \geq \rho(H)$. This process is repeated until there is no remaining removable blocks. Finally, the algorithm returns a subgraph H having the largest check-in density

Time complexity. It takes $O(|V_U|\,(|V_U|+|E_U|)+|V_L|(|V_L|+|E_L|))$ to generate the blocks in both networks, where $(|V|+|E|)$ is for traversing a graph and $|V|$ is the number of blocks. The maximal number of iteration is $|V_U|+|V_L|$. Note that in each iteration, we need to regenerate all blocks. It takes $O(|E_U|)$ and $O(|E_L|)$ to find k-core for both networks. Thus, the complexity of the algorithm is $O((|V_U|+|V_L|)(|V_U|(|V_U|+|E_U|)+|V_L|(|V_L|+|E_L|)))$.

5 Expansion Algorithm

When a graph is very large, GRA may cost long running time. We next propose a heuristic expansion algorithm.

5.1 Greedy Expansion Algorithm (GEA)

The main idea of our expansion algorithm is to start from the query nodes, and then gradually expand the size of the current solution, by adding nodes iteratively to the current solution based on certain greedy criteria.

In this subsection we introduce a simplied version of our expansion algorithm. We intialize the initial current solution with the set of query nodes. The algorithm maintains a priority queue that stores candidate nodes to be inserted into the current solution. In the initial stage, the priority queue contains neighbor nodes of query nodes. Initially, if the social network (resp. the location network) does not contain any query node, a neighbor node in check-in network of the current solution will be inserted into the priority queue. If query nodes are in both networks, neighbor nodes of query nodes are inserted into the priority queue. The algorithm then iteratively expands the current solution as follows: In each iteration, it removes the node with the highest priority from the priority queue and includes it into the current solution; Then all the neighbor nodes of the removed node in both networks are inserted into the priority queue; Finally, we need to update the priority of neighbor nodes of the removed node in the priority queue. At the end of each iteration, we check if the current solution satisfies the cohesiveness constraint: if yes, we terminate the algorithm and return the current solution as the result. For checking the connectivity constraint of a graph, we use Euler tour tree [16].

A key problem in the expansion algorithm is how to set the priority for the priority queue. We design the following three criteria to sort nodes in the priority queue:

- CR 1. sum of check-in edge weights to the current solution.
- CR 2. number of neighbor nodes that are in the current solution but do not satisfy the cohesiveness constraint.
- CR 3. number of neighbor nodes in the current solution.

The first criterion is to give higher priority to nodes with larger check-in edge weight since they can increase the check-in density of the current solution. The ties are broken based on criteria 2 and 3 in order. The second criterion is to give higher priority to nodes that have more neighbor nodes in the current solution that do not satisfy the cohensiveness constraint. This is because including such nodes in the current solution will increase the chance of the current solution to meet the cohensiveness constraint. The third criterion is to give higher priority to nodes with a higher number of neighbor nodes in the current solution as such nodes contribute more to the cohensivness of the result.

5.2 Advanced Greedy Expansion Algorithm(GEA$^+$)

The simplfied expansion algorithm (referred to as GEA) (1) terminates only when the cohesiveness constraints are satisfied, and (2) till then it iteratively includes neighboring nodes that have higher check-in density into the current solution according to the CR 1. This might end up reporting a low-quality solution (i.e., with low check-in density) as we explain below. Consider the following scenario that the neighbor nodes of a query node from the same network must be included in the solution to satisfy the cohesiveness constraints; however they have very few check-in edges with the other network. In GEA, such neighbor nodes have to be included in the current solution until the cohesiveness constraints are satisfied. Thus, the size of the final solution which satisfies the cohesiveness constraints may become relatively large, and the overall check-in density gets reduced. To solve

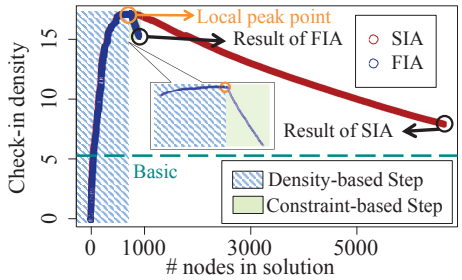

Figure 4: Check-in density plot of GEA and GEA$^+$

this problem, we propose our advanced greedy expansion algorithm, referred to as GEA$^+$. The key idea of GEA$^+$ is to change the sorting criteria for the priority queue when a *local peak point* in the check-in density is reached for the first time. The *local peak point* is defined as follows.

DEFINITION 10. *[Local peak point]. Given a series of check-in graphs $T_1 \subset T_2 \subset ... \subset T_k$ that are the intermediate results reported by GEA$^+$, and a sliding window size w, a local peak point is an intermediate check-in graph T_i such that $\rho(T_i) \geq \rho(T_{i-w'})$ and $\rho(T_i) \geq \rho(T_{i+w''})$ where $w = w' + w'' + 1$ and $\sum_{i=1}^{w-1}(\rho(T_{i+1}) - \rho(T_i)) < 0$.*

Figure 4 shows an example of the variation of check-in density over the solution size in GEA and GEA$^+$. The orange circle is a local peak point. At this point, the priority criteria used in GEA$^+$ are changed in order to satisfy the cohesiveness constraints quickly — without significantly reducing the check-in density, compared to that at the local peak point. In contrast, GEA would include more nodes in the solution to satisfy the cohesiveness constraints, which greatly reduces the check-in density.

GEA algorithm works in two steps as introduced next.

Density-based expansion step: The density-based expansion step is the same as GEA. However, the step is to find a local peak point instead of finding the final solution in GEA.

Constraint-based expansion step: Once a local peak point is reached, we would like the current solution to satisfy the cohesiveness constraints by adding as few nodes as possible, without significantly reducing the check-in density compared to that at the local peak point. To achieive this goal, we change the sorting order for priority in the priority queue—We rearrange all the remaining nodes in the priority queue in the order of CR 2 > CR 3 > CR 1. In this way, we can quickly include the nodes in the current solution so that the cohesiveness constraints are satisfied.

To find a solution with better check-in density, GEA$^+$ algorithm may keep searching the next solutions. If the check-in density is increased in the next solution, the process continues. This algorithm terminates when the next solution has smaller check-in density compared to the previous solution.

Time complexity. GEA$^+$ takes $O(|V_U| + |E_U| + |V_L| + |E_L|)$ time for traversing both networks and $O((|E_C| + |V_U| + |V_L|)log(|V_U| + |V_L|))$ time for maintaining the priority queue. It takes $O(|E| \log |V|)$ to setup the Euler tour tree for checking the graph connectivity. It takes $O(|V||Q| \log |V|)$ for the connectivity test of query nodes when $|Q_U| \geq 2$ or $|Q_L| \geq 2$. Thus, the complexity of GEA$^+$ is $O((|E_C|+|V_U|+|V_L|)log(|V_U|+|V_L|)+|V_U||Q_U| \log |V_U| + |V_L| |Q_L| \log |V_L| + T_{BA})$. Note that time complexity of GEA$^+$ algorithm is the same as that of GEA. Clearly, GEA$^+$ has a better time complexity than GRA.

Exploring multiple local peak points. Instead of one local peak point, we can explore multiple local peak points to get a better solution. Specifically, after we get the geosocial community C from the first local peak point L_1, we continue the density-based expansion from L_1 to reach the second local peak point L_2. If $\rho(L_2) > \rho(C)$, we conduct the constraint-based expansion from L_2 and get a new geosocial community C'. If C' has a higher check-in density, we update the result as C'. Then we continue the density-based expansion from L_2 to get the third local peak point. We repeat these steps for every local peak points.

6 Experiments

We evaluate proposed algorithms over real-world location-based social networks (LBSNs). All experiments were conducted on a server running Ubuntu 16.04 with 64GB memory and 2.60GHz Xeon CPU E5-4627 v4.

6.1 Experimental setup

6.1.1 Datasets. Table 3 shows the characteristics of datasets that we used in our experiments. The largest network (*Gowalla*) has 1.4 million nodes.

Table 3: Dataset characteristics

Dataset		$\|U\|$	$\|S\|$	#check-ins
BK	Brightkite [8]	58,228	772,966	4,747,281
GL	Gowalla [8]	196,591	1,280,969	6,442,890
YP	Yelp [24]	22,917	18,955	860,888
TW	Twitter [23]	554,372	534,749	554,372

6.1.2 Parameter setting. The default parameter values and their variations are given in Table 4. The default sliding window size w is set as 100, to decide local peak points in our expansion algorithm.

Query node generation To obtain query nodes Q, we use the personalized PageRank algorithm [20, 25] in a merged network $G_T = (V_U[K_U] \cup V_L[K_L], E_C[K_U \cup K_L] \cup E_U[K_U] \cup E_L[K_L])$ where K_U and K_L are the result of k-core decomposition in social and location networks with minimum degree threshold m. We randomly select a query node in G_T to run personalized PageRank. The result PPR score indicates the importance for the important nodes in the vicinity of that query node. We select top-$|Q|$ nodes with the highest PPR scores as query nodes Q.

6.1.3 Algorithms. To the best of our knowledge, our problem does not have no direct competitor in the literature. Thus, we compare the proposed algorithms in our experiments.

6.2 Experimental results

Overall performance on different datasets We first study the performance of our three algorithms on the four datasets. Here, we set query size $|Q| = 2$, $m = 20$, and $r = 0.1$. One query node is from social network and the other is from

Table 4: Parameters with the default values underlined

Variable	Variation	Description
r	0.1,<u>0.2</u>,0.3,0.4,0.5	max. spatial edge length (km)
m	<u>5</u>,10,15,20	min. degree threshold
$\|Q\|$	<u>1</u>,2,4,8,16,32	number of query nodes

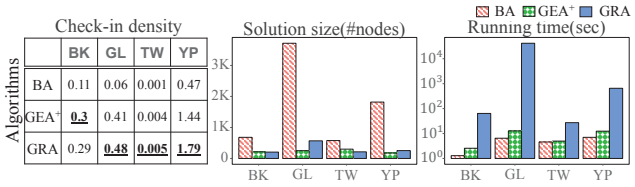

Figure 5: Check-in density, solution size, and running time for four LBSNs

location network. For each case, we report the average value over 5 different queries.

Clearly, our greedy removing algorithm GRA achieves the highest check-in density in GL, TW, and YP, while GEA$^+$ shows the highest check-in density on BK. We also observe that our advanced expansion algorithm GEA$^+$ acheives comparable quality in terms of check-in density. However, in terms of running time, GEA$^+$ is more than five orders of magnitude faster than GRA. BA has the worst quality, although it is very efficient. In terms of solution size, BA returns much larger subgraphs than GEA$^+$ and GRA since BA directly uses the result of core decomposition for finding a solution as we discussed in Section 3.3.

Varying number of query nodes $|Q|$ This set of experiments is to study the effect of query size $|Q|$ on performance. Figure 6 shows the the result on YP as we vary the number of query nodes $|Q|$. We oberve similar trends as we do in the last experiments: GEA$^+$ achieves comparable check-in density compared to GRA, while GEA$^+$ is several orders of magnitude faster than GRA; BA performs poorly in terms of quality although it runs fast. We also find that the check-in density, solution size, and running time are not very sensitive with respect to the number of query nodes. This is perhaps because all query nodes are nearby for a query set.

Scalability results To demonstrate the scalability of our algorithms, we consider subgraphs of different sizes by changing the maximum spatial radius r from the original LBSNs. Figure 7 shows the running time of the three algorithms. Note that GRA fails to finish on the large datasets GL within 24 hours when maximum spatial edge length $r \geq 0.2$. We observe that the running times of the BA and GEA$^+$ are very close to each other, and increase almost linearly with the LBSN size. Note that the running time of GEA$^+$ and GRA highly depends on the size of BA.

Effect of parameters We first study the effect of the maximum spatial edge length r on performance. Note that in this

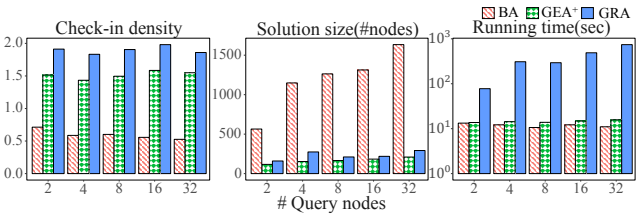

Figure 6: Varying $|Q|$ on YP ($m = 20$)

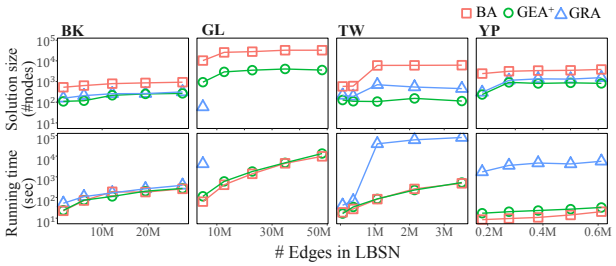

Figure 7: Scalability analysis by varying LBSN size

experiment we use the default values for other parameters; We do not show GRA, because it does not finish within 24 hours in most cases. Figure 8 shows that as we increase r, the check-in density mostly becomes better, and the size of the solution becomes larger. This is because in location graph, the number of candidates for a cluster increases as the number of edges and nodes having at least m core index increases. We observe that as spatial edge length r increases, the running time of both algorithms increases due to the increased number of edges in the location graph. We notice that the solution size of BA increases significantly as we increase r. This is because as the search space increases, more nodes can be considered, which produces larger solution due to the characteristics of the basic algorithm that finds relatively large results.

We next study the effect of the minimum degree threshold m on performance. Figure 9 shows the result. As we increase m, the internal density of both social subgraph and location subgraph in the solution becomes larger. However, we find that the check-in density decreases with larger m, because high internal density in user side and location side does not ensure a high check-in density across them. As expected, when m increases, the running time decreases and the solution size becomes smaller since many nodes do not satisfy the degree constraint and are disregarded.

Figure 5 shows the check-in density, solution size, and running time of the three algorithms.

Usefulness of BA and comparing GEA and GEA$^+$ Figure 10 shows the result of comparing the algorithms without using BA, and algorithms with BA which is denoted by subscript 'b' in Figure 10 (on the YP dataset). In the case of expansion algorithms, using the result of BA consistently gives better result in terms of the check-in density, solution

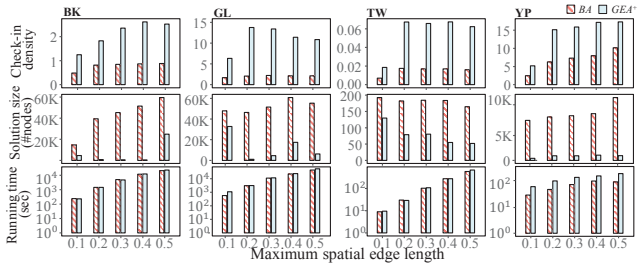

Figure 8: Varying maximum spatial edge length, $|Q|$=1

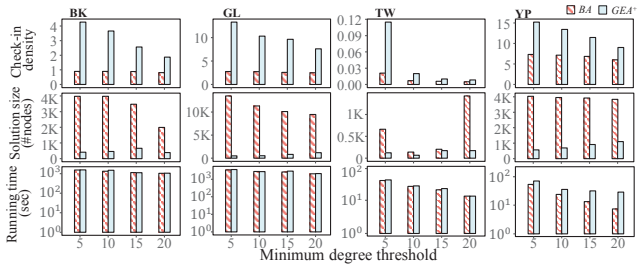

Figure 9: Varying minimum degree threshold, |Q|=1

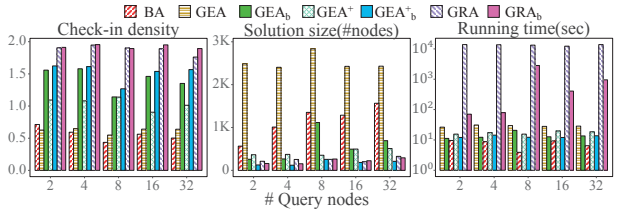

Figure 10: With BA vs without BA

size, and the running time than the expansion algorithms without using BA. In the case of GRA, there is no much difference in the check-in density. However, using the result of BA can greatly reduce execution time.

Summary GRA and GEA$^+$ achieve much better check-in density than BA. GEA$^+$ returns solutions with comparable quality with GRA while GEA$^+$ is faster than GRA by orders of magnitude. BA returns a result of large size with low density, though its running time is comparable with that of GEA$^+$.

6.3 Case study

We conduct two case studies to show the usefulness of GCS. We use GEA$^+$ for the case study and set m=3, r=0.1km.

Event Organization We consider two scenarios. First, we help a user organize a party at a specific venue by recommending a group of people to invite. Intuitively, the invitees should be densely connected to be engaged in the party. Moreover, as people prefer to go to nearby locations for different purposes [22], it would be better if the invitees frequently visit locations near the venue. To find such invitees, we issue a GCS query, which consists of the organizer (green circle) and the venue location (green rectangle). The set of users returned by the GCS problem is shown in Figure 11 CASE #1. We observe that GCS finds a community with 52 users and the check-in density is 4.17.

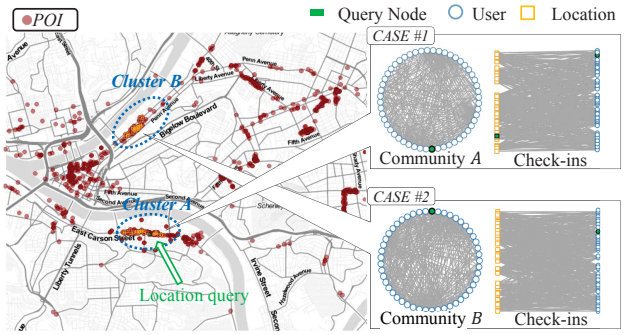

Figure 11: Two location clusters and their corresponding user communities in Pittsburgh (YP dataset). The users in the Community, locations in the cluster, as well as those identified users and locations via the check-in edges are all densely connected.

Second, we still help the user organize a party, but the venue for the party is undecided yet. Hence, we would like to recommend both a group of people to invite and a set of candidate venues. We use the same organizer (green circle) as the GCS query node. GEA$^+$ algorithm returns a location cluster B (highlighted in orange color) with 48 locations and a user community B with 49 nodes as the result as shown in Figure 11. The cluster B contains Cafe, French Restaurants, Dance club, Bar. We observe that different from the first application, the new GCS query recommends a different user community and a location cluster.

7 Conclusions

In this paper, we formulated the GeoSocial Community Search problem (GCS) that aims to find a social community and a cluster of spatial locations that are densely connected in location-based social networks. We proved that the problem is NP-hard and is not in APX. We proposed three solutions to the problem. Extensive empirical studies on large-scale real-world location-based social networks demonstrate the efficiency and effectiveness of the proposed algorithms.

Acknowledgments

Gao Cong is partially supported by Singtel Cognitive and Artificial Intelligence Lab for Enterprises (SCALE@NTU), which is a collaboration between Singapore Telecommunications Limited (Singtel) and Nanyang Technological University (NTU) that is funded by the Singapore Government through the Industry Alignment Fund - Industry Collaboration Projects Grant, a MOE Tier-2 grant MOE2016-T2-1-137, and a NTU ACE grant. Arijit Khan is supported by MOE Tier-1 grant 2019-T1-002-059.

References

[1] Omid Amini, David Peleg, Stéphane Pérennes, Ignasi Sau, and Saket Saurabh. 2008. Degree-constrained subgraph problems: Hardness and approximation results. In *International Workshop on Approximation and Online Algorithms*. Springer, 29–42.

[2] Omid Amini, David Peleg, Stéphane Pérennes, Ignasi Sau, and Saket Saurabh. 2012. On the approximability of some degree-constrained subgraph problems. *Discrete Applied Mathematics* 160, 12 (2012), 1661–1679.

[3] Nicola Barbieri, Francesco Bonchi, Edoardo Galimberti, and Francesco Gullo. 2015. Efficient and effective community search. *Data Mining and Knowledge Discovery* 29, 5 (2015), 1406–1433.

[4] Vladimir Batagelj and Matjaz Zaversnik. 2003. An O (m) algorithm for cores decomposition of networks. *arXiv preprint cs/0310049* (2003).

[5] V. Batagelj and M. Zaversnik. 2011. Fast Algorithms for Determining (Generalized) Core Groups in Social Networks. *Advances in Data Analysis and Classification* 5, 2 (2011), 129–145.

[6] Moses Charikar. 2000. Greedy approximation algorithms for finding dense components in a graph. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*. Springer, 84–95.

[7] Lu Chen, Chengfei Liu, Rui Zhou, Jianxin Li, Xiaochun Yang, and Bin Wang. 2018. Maximum Co-located Community Search in Large Scale Social Networks. *Proceedings of the VLDB Endowment* 11, 9 (2018).

[8] Eunjoon Cho, Seth A Myers, and Jure Leskovec. 2011. Friendship and mobility: user movement in location-based social networks. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 1082–1090.

[9] Pierluigi Crescenzi. 1997. A short guide to approximation preserving reductions. In *Proceedings of Computational Complexity. Twelfth Annual IEEE Conference*. IEEE, 262–273.

[10] Wanyun Cui, Yanghua Xiao, Haixun Wang, and Wei Wang. 2014. Local search of communities in large graphs. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 991–1002.

[11] Yixiang Fang, Reynold Cheng, Xiaodong Li, Siqiang Luo, and Jiafeng Hu. 2017. Effective community search over large spatial graphs. *Proceedings of the VLDB Endowment* 10, 6 (2017), 709–720.

[12] Yixiang Fang, Reynold Cheng, Siqiang Luo, and Jiafeng Hu. 2016. Effective community search for large attributed graphs. *Proceedings of the VLDB Endowment* 9, 12 (2016), 1233–1244.

[13] Shanshan Feng, Gao Cong, Bo An, and Yeow Meng Chee. 2017. POI2Vec: Geographical Latent Representation for Predicting Future Visitors. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*. AAAI Press, 102–108.

[14] Antonin Guttman. 1984. *R-trees: a dynamic index structure for spatial searching*. Vol. 14. ACM.

[15] Samiul Hasan and Satish V Ukkusuri. 2015. Location contexts of user check-ins to model urban geo life-style patterns. *PloS one* 10, 5 (2015), e0124819.

[16] Monika R Henzinger, Valerie King, and Valerie King. 1999. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *Journal of the ACM (JACM)* 46, 4 (1999), 502–516.

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

[18] Xin Huang and Laks VS Lakshmanan. 2017. Attribute-driven community search. *Proceedings of the VLDB Endowment* 10, 9 (2017), 949–960.

[19] Xin Huang, Laks VS Lakshmanan, Jeffrey Xu Yu, and Hong Cheng. 2015. Approximate closest community search in networks. *Proceedings of the VLDB Endowment* 9, 4 (2015), 276–287.

[20] Glen Jeh and Jennifer Widom. 2003. Scaling personalized web search. In *Proceedings of the 12th international conference on World Wide Web*. Acm, 271–279.

[21] Samir Khuller and Barna Saha. 2009. On finding dense subgraphs. In *International Colloquium on Automata, Languages, and Programming*. Springer, 597–608.

[22] Justin J Levandoski, Mohamed Sarwat, Ahmed Eldawy, and Mohamed F Mokbel. 2012. Lars: A location-aware recommender system. In *2012 IEEE 28th international conference on data engineering*. IEEE, 450–461.

[23] Guoliang Li, Shuo Chen, Jianhua Feng, Kian-lee Tan, and Wen-syan Li. 2014. Efficient location-aware influence maximization. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. ACM, 87–98.

[24] Yiding Liu, Tuan-Anh Nguyen Pham, Gao Cong, and Quan Yuan. 2017. An experimental evaluation of point-of-interest recommendation in location-based social networks. *Proceedings of the VLDB Endowment* 10, 10 (2017), 1010–1021.

[25] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank citation ranking: Bringing order to the web.* Technical Report. Stanford InfoLab.

[26] Tuan-Anh Nguyen Pham, Xutao Li, Gao Cong, and Zhenjie Zhang. 2015. A general graph-based model for recommendation in event-based social networks. In *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015*. IEEE Computer Society, 567–578.

[27] Stephen B Seidman. 1983. Network structure and minimum degree. *Social networks* 5, 3 (1983), 269–287.

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

[29] Kai Wang, Xin Cao, Xuemin Lin, Wenjie Zhang, and Lu Qin. 2018. Efficient computing of radius-bounded k-cores. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 233–244.

[30] Fan Zhang, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin. 2017. When engagement meets similarity: efficient (k, r)-core computation on social networks. *Proceedings of the VLDB Endowment* 10, 10 (2017), 998–1009.