Maximum Co-located Community Search in Large Scale Social Networks

Lu Chen[†], Chengfei Liu[†], Rui Zhou[†], Jianxin Li[¶], Xiaochun Yang[§], Bin Wang[§]
[†]Swinburne University of Technology, [¶]University of Western Australia, [§]Northeastern University
[†]{luchen, cliu, rzhou}@swin.edu.au [¶]jianxin.li@uwa.edu.au
[§]{yangxc, binwang}@mail.neu.edu.cn

ABSTRACT

The problem of k-truss search has been well defined and investigated to find the highly correlated user groups in social networks. But there is no previous study to consider the constraint of users' spatial information in k-truss search, denoted as *co-located community search* in this paper. The co-located community can serve many real applications. To search the maximum co-located communities efficiently, we first develop an efficient exact algorithm with several pruning techniques. After that, we further develop an approximation algorithm with adjustable accuracy guarantees and explore more effective pruning rules, which can reduce the computational cost significantly. To accelerate the real-time efficiency, we also devise a novel quadtree based index to support the efficient retrieval of users in a region and optimise the search regions with regards to the given query region. Finally, we verify the performance of our proposed algorithms and index using five real datasets.

PVLDB Reference Format:
Lu Chen, Chengfei Liu, Rui Zhou, Jianxin Li, Xiaochun Yang, and Bin Wang. Maximum Co-located Community Search in Large Scale Social Networks. *PVLDB*, 11 (10): 1233-1246, 2018.
DOI: https://doi.org/10.14778/3231751.3231755

1. INTRODUCTION

With the increasing popularity of online social networks, one of the most important tasks in social network data analytics is to find communities of users with close structural connections each other. The extensive studies on finding communities can be categorized into *global community detection GCD* (e.g., [19, 20, 21, 34, 16, 6]), *local community detection LCD* (e.g., [14, 43]), *global community search GCS* (e.g., [31, 30, 35]), and *local community search LCS* (e.g., [18, 39, 14, 13, 24, 17, 44, 23]). Community detection methods are often used to discover communities in social networks based on the predefined implicit criteria, e.g., modularity [19]. The main difference between GCD and LCD is that each user is equivalently important to be measured in GCD, while the importance of a user depends on his relevance to the given query vertex in LCD. Different from community detection, the community search methods concentrate on finding communities from social networks based on the users' specified explicit criteria, e.g., parameter k in k-core based model [38], k-truss based model [12], and k-edge-connected component based model [4]. Similar to community detection methods, the major difference between GCS and LCS is that LCS requires the communities to contain the given query vertex, but GCS doesn't have such additional requirement. However, most works above didn't consider the effect of users' spatial information in their community detection or search methods.

Proceedings of the VLDB Endowment, Vol. 11, No. 10
Copyright 2018 VLDB Endowment 2150-8097/18/06.
DOI: https://doi.org/10.14778/3231751.3231755

Searching communities with social and spatial cohesiveness is of great importance in many applications, e.g., event scheduling, product recommendation, targeted advertisement, local activism and advocacy, as well as more effective content spreading like shop promotions, local news, and job openings. Although spatial feature is highly desirable in applications, in practice, the existing study on spatial social community is still limited. In [17], Fang et al. require all the vertices of a returned k-core community in a minimum covering circle with the smallest radius and the resultant community must contain the given query vertex. So it is a type of LCS with the spatial constraint. In [16, 6], Expert et al. and Chen et al. take into account spatial information in the process of community detection by weighting the link based on the spatial distance of two linked users. It is a type of GCD with the spatial constraint. However, the two types of work cannot guarantee the spatial closeness of the community members, which will be further discussed in our experiments. In [45], Zhang et al. require all the vertices of a returned k-core based community meeting similarity constraints, where the similarity could be distance similarity. However, finding exact result for this community model is expensive in large scale social network since its NP-hardness.

Therefore, in this paper, we investigate the *co-located community* search problem that reveals the maximum communities with high social and spatial cohesiveness, denoted as (k,d)-MCCs search. The social cohesiveness is defined using the minimum truss value k [12] and the spatial cohesiveness is parameterised by a user-specified distance value d. As such, our proposed (k,d)-MCCs search problem can allow users to easily affirm the quality of the resultant communities, which also fills in the research gap on the type of GCS with spatial constraint.

Given a social network G and two parameters k and d, a straightforward approach is to enumerate all possible subgraphs in G meeting minimum truss value k where the number of the subgraphs could be as large as $\mathcal{O}(2^n)$. It then filters the candidates having a node pair with their distance above the spatial closeness threshold d. So the time complexity of this approach is at least $\mathcal{O}(2^n)$ where n is the number of vertices in G. Obviously, it is infeasible to use this approach to support online (k,d)-MCCs search, particularly for the large scale social networks. Thus, this paper focuses on devising efficient algorithms to achieve real-time response with theoretical guarantee.

To address the challenge of efficiency, we first develop an exact (k,d)-MCCs search algorithm by proposing novel pruning techniques. During the search, we explore techniques to prune the search space significantly by considering upper bound based earlier termination, heuristic search order, and conditions for reusing pruning computation. Before searching, we also propose pre-pruning techniques for reducing magnitudes of input data. To design polynomial algorithms, we develop a novel approximation schema with spatial accuracy guarantees. Notice, our proposed approximation scheme can provide adjustable spatial error ratios based on user's requirement on the spatial accuracy. To further improve the performance of the approximation algorithm, we propose more pruning techniques and also design the novel index TQ-tree. The main contributions of our work are summarised as follows.

- We propose a novel *co-located community* model and formally define the (k,d)-MCCs search problem. (Section 2)
- We develop an efficient exact algorithm for finding (k,d)-MCCs by proposing effective techniques for pruning before and during the search. (Section 3)
- We also develop a spatial approximation algorithm that offers a variable spatial error ratio ranging from $2\sqrt{2}+\epsilon$ to $\sqrt{2}+\epsilon'$. The efficiency of the approximation algorithm is further improved by proposing more effective pruning techniques and a novel TQ-tree index. (Section 4)
- We conduct extensive experimental studies on five real datasets to demonstrate the efficiency and effectiveness of the proposed algorithms. (Section 5)

2. PROBLEM DEFINITION

We consider a social network graph $G = (V, E)$, which is an undirected graph with vertex set $V(G)$ and edge set $E(G)$, where vertices represent social users and edges denote their friendships. For each vertex $v \in V(G)$, it has a spatial attribute $(v.x, v.y)$, where $v.x$ and $v.y$ denote its spatial positions along $x-$ and $y-$axis in a two-dimensional space.

Co-located community. A co-located community is a subgraph $J \subseteq G$ satisfying: (1) connectivity: J is connected, (2) structural cohesiveness: all vertices in J are connected intensively, and (3) spatial cohesiveness: all vertices in J are spatially close with each other.

Structural cohesiveness. We consider truss as the metric to measure the structural cohesiveness of a co-located community. Truss measures the number of triangles that each edge is involved in a graph. Given J, let us denote a triangle involving vertices $u, v, w \in V(J)$ as $\triangle_{uvw}$. The support of an edge $e(u,v) \in E(J)$, denoted by $sup(e, J)$, is the number of triangles containing e, i.e., $sup(e,J) = |\{\triangle_{uvw} : w \in N(v,J) \cap N(u,J)\}|$, where $N(v,J)$ and $N(u,J)$ are the neighbours of v, u in J correspondingly. Next, we define the truss of a co-located community J as follows:

Definition 1. Subgraph truss. *The truss of $J \subseteq G$, where $|V(J)| \geq 2$, is the minimum support of an edge in J plus 2, i.e., $\tau(J) = 2 + \min_{e \in E(J)}\{sup(e,J)\}$.*

J is a connected k-truss if it is both *connected* and $\tau(J) \geq k$. Intuitively, a k-truss is subgraph in which each connection (edge) (u,v) has at least $k-2$ common neighbours. A k-truss with a large value k indicates strong internal connections over members. In a k-truss, each node should have degree at least $k-1$, implying a k-truss must be a $(k-1)$-core. A connected k-truss is also $(k-1)$-edge-connected.

Spatial cohesiveness. Let $ed(u,v)$ denote the spatial distance between vertices u and v. We first introduce the concept of spatial co-location to measure the spatial cohesiveness. Then we define the co-located community formally.

Definition 2. Spatial co-location. *Given a distance threshold d, a subgraph $J \subseteq G$ is a spatial co-location graph if for every pair $u, v \in V(J)$, $ed(u,v) \leq d$ holds.*

Definition 3. Co-located community. *Given a graph G, a positive integer k, and a spatial distance d, J is a co-located community, if J satisfies the following constraints:*
- ***Structural cohesiveness**. J is connected, $\tau(J) \geq k$.*
- ***Spatial cohesiveness**. J is a spatial co-location graph w.r.t. a spatial distance d.*

In general, when searching a community, users may want to maximise the members contained in the community once they fix the spatial and social cohesiveness parameters. Therefore, in this paper, given a graph G, we study finding the maximum co-located communities, denoted as (k,d)-MCCs where k stands for k-truss, d for spatial distance, M for maximum and CC for co-located community. Now we formally define the problem of (k,d)-MCCs search.

Problem 1. (k,d)-MCCs search. *Given a graph G, positive integer k and number d, return any of those maximum co-located communities $J \subseteq G$, satisfying constraints:*
- *J is a co-located community.*
- *There is no another co-located community J' such that $|V(J')| > |V(J)|$.*

For example, in Figure 1(b), vertices in dark blue coloured areas are co-located. Similarly, in Figure 1(a), three possible co-located communities are in blue coloured areas with $k = 4$. The (4,d)-MCC here is the subgraph containing vertices $\{d, e, f, g, h, i\}$ with cardinality 6, as it is the maximum.

We may find (k,d)-MCCs from G by inspecting the whole graph. However, to improve the search performance, we only want to search the parts of G that may contain (k,d)-MCCs. To achieve that, we introduce the theorem below:

Theorem 1. (k,d)-MCC*s of a graph G can be found from one of the maximal connected k-trusses of G if they exist.*

The proof is trivial since vertices that are not part of a maximal connected k-truss clearly cannot meet the structural cohesiveness requirement in Definition 3.

By Theorem 1, the intuitive steps to find (k,d)-MCCs in G include: (1) compute maximal connected k-trusses (note: these k-trusses are non-overlapped), (2) search the local (k,d)-MCCs in each of these k-trusses, and (3) find the global (k,d)-MCCs from the locals by comparing the cardinalities.

Analysis. A k-truss index can be built within $\mathcal{O}(|E(G)|^{\frac{3}{2}})$ for a graph G. The k-truss index for G is essentially a list of

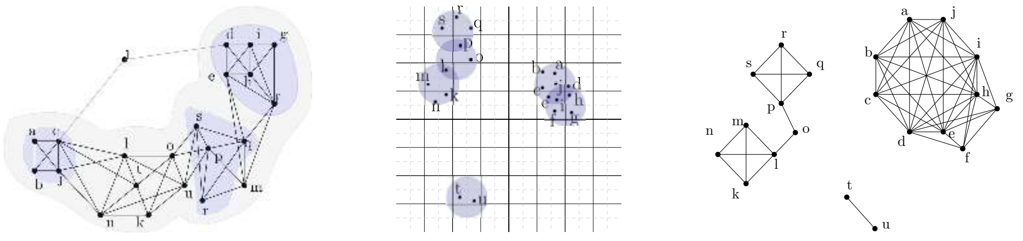

(a) Graph data (b) Spatial DIST (c) Spatial network

Figure 1: Spatial attributed graph

Table 1: Notations

Notation	Definition
T	initially a maximal connected k-truss graph
T'	spatial neighbourhood network for T
T'_0	a connected component of T', $T'_0 \subseteq T'$
u, v, w	individual vertices
$ed(u, v)$	spatial distance between u, v
$gd(u, v, T)$	distance between u, v in T
$deg(u, G)$	degree of u in G
$N(u, G)$	neighbours of u in G
$\tau(G)$	the minimum truss of G
$A, \mathcal{A}$	a maximal clique, a set of maximal cliques
R, P, X	vertices sets
$T(R), T'(R)$	subgraphs of T and T' induced by vertices in R
c	a square spatial space cell with width w
m, M	a landmark cell and a set of landmark cells
r	a square spatial region consisting of cells
ζ	an integer, denoting a number of cells
V_r	a set of vertices located in a region r
p	an error-bounded search bound region
$\mathcal{K}(r)$	the k-truss in a region r

edges associated with their *edge trusses* defined by $\tau(e, G) = max_{H \subseteq G \wedge e \in E(H)} \{\tau(H)\}$ [24]. With the k-truss index, given a k, we can retrieve all maximal connected k-trusses in G in polynomial time. However, it is still challenging to find local (k,d)-MCCs within a maximal connected k-truss due to: (1) the total number of spatial co-location subgraphs in the k-truss could be exponential [22] and (2) there is no guaranteed monotonic relationship between the size of a co-location subgraph and the size of its co-located communities.

3. FINDING EXACT RESULTS

We first introduce a definition as follows:

Definition 4. Spatial neighbourhood network. *Given a T and a distance d, a spatial neighbourhood network for T is a graph T', which is an undirected graph with $V(T')$=$V(T)$ and $E(T')$=$\{(u, v) | ed(u, v) \leq d \ \wedge \ u, v \in V(T)\}$.*

Finding a (k,d)-MCC is equivalent to finding an unextendable vertex set R such that the R-induced subgraph $T'(R)$ of T' is a clique while the R-induced subgraph $T(R)$ of T contains a connected k-truss $G_R = (R, E_R)$ where $E_R \subseteq E(T(R))$.

Next, we show the baseline algorithm to find (k,d)-MCCs.

3.1 Baseline Algorithm

Given T and T', the baseline algorithm is to find all the maximal cliques contained by T', and check the sorted maximal cliques one by one. For each maximal clique A (the set of vertices in a maximal clique), we need to compute local (k,d)-MCCs in $T(A)$. After all maximal cliques have been checked, we compare the cardinalities of the local (k,d)-MCCs and get the global (k,d)-MCCs.

Algorithm 1: baseline(T,T', $b = 0$)

```
1  $\mathcal{R} \leftarrow$mccSearch($T, T', b$);
2  Return $\mathcal{R}$;
3  Procedure mccSearch($T, T', b$)
4      $\mathcal{A} \leftarrow$ bkp($\emptyset, V(T'), \emptyset$);
5      sort $\mathcal{A}$ in descending order by clique cardinality;
6      for each $A \in \mathcal{A}$ do
7          if $|A| > b$ then
8              $\mathcal{R}' \leftarrow$ maximum $k$-trusses in $T(A)$ ;
9              $b \leftarrow$ kdmccCollect($\mathcal{R}, \mathcal{R}', b$);
10 Procedure kdmccCollect($\mathcal{R}, \mathcal{R}', b$)
11     if $|V(\mathcal{R}'[0])| == b$ then
12         collect $\mathcal{R}'$ into $\mathcal{R}$;
13     if $|V(\mathcal{R}'[0])| > b$ then
14         $b \leftarrow |V(\mathcal{R}'[0])|$;
15         replace $\mathcal{R}$ by $\mathcal{R}'$;
```

Baseline algorithm. The baseline algorithm is presented in Algorithm 1. It ensures the correctness by giving every maximal clique $A \in \mathcal{A}$ a chance. To improve the search efficiency, Algorithm 1 uses a heuristic rule and a bound to prune small maximal spatial cliques. The heuristic rule assumes the larger the size of a spatial clique is, the larger the size of the contained (k,d)-MCCs may be. The heuristic rule is implemented by sorting the generated maximal cliques (line 5). The bound b is initialised as 0 and is continuously updated as the maximum size of the (k,d)-MCCs found so far. A maximal clique is pruned if its size is less than b.

Collect candidate results. In Algorithm 1, **Procedure** kdmccCollect is used to collect candidate results. It checks the maximality of the currently found (k,d)-MCCs in $\mathcal{R}'$ and determines if they should be added into previously found results in $\mathcal{R}$, or replace $\mathcal{R}$, or be discarded (lines 12 to 16 in Algorithm 1). During the process, the upper bound will be updated if necessary.

Avoid duplication. Since (k,d)-MCCs are contained by spatial-clique-induced subgraphs of T, it is possible that multiple spatial cliques contain the same (k,d)-MCC. To avoid duplication, we assign a unique key to each (k,d)-MCC based on the vertices it contains. Before a new (k,d)-MCC is collected into the result $\mathcal{R}$, duplication will be checked by verifying if its key has already existed.

Example. We show an example using Algorithm 1 to find (4,d)-MCCs. The input social graph is the 4-truss in Figure 1(a) and its spatial network is in Figure 1(c). Firstly,

Table 2: Maximal cliques contained in Figure 1(c)

Cad.	Cliques	Cad.	Cliques
8	$\{a, b, c, d, e, h, i, j\}$	6	$\{d, e, f, g, h, i\}$
4	$\{r, s, p, q\}$, $\{m, n, k, l\}$	2	$\{p, o\}$, $\{o, l\}$,$\{t, u\}$

Table 3: Enumeration trace

Iter.	clique	bound	$\mathcal{R}$
0	NULL	$b = 0$	$\emptyset$
1	$\{a, b, c, d, e, h, i, j\}$	$b = 4$	$\{\{a, c, b, j\}\}$ $\{\{d, e, h, i\}\}$
2	$\{d, e, f, g, h, i\}$	$b = 6$	$\{\{d, e, h, g, f, i\}\}$
3	$\{r, s, p, q\}$	$b = 6$	$\{\{d, e, h, g, f, i\}\}$

maximal cliques can be obtained and sorted by size (see Table 2). The upper bound history and the corresponding (k,d)-MCCs after each iteration are displayed in Table 3. The iteration stops when the upper bound $b = 6$ is larger than the sizes of the remaining cliques.

Time complexity. The dominating part of Algorithm 1 is to list all maximal cliques that would be $\mathcal{O}(3^{\frac{|V(T)|}{3}})$, using algorithm bkp in [40]. Another part is to find k-trusses with maximum cardinality in $T(A)$ where A is the vertex set contained by a maximal clique of T'. To compute the maximum k-trusses in $T(A)$, we use the method in [41] bounded by $\mathcal{O}(|E(T)|^{\frac{3}{2}})$. Therefore, the complexity of Algorithm 1 is $\mathcal{O}(3^{\frac{|V(T)|}{3}} + \sum_{A\in\mathcal{A}} |E(T(A))|^{\frac{3}{2}})$, where $\mathcal{A}$ is the set of maximal spatial cliques contained by T'.

3.2 Efficient (k,d)-MCC Search

The baseline method finds (k,d)-MCCs in two steps. Firstly, it generates all vertex sets meeting the requirement of spatial cohesiveness, i.e. spatial cliques. Secondly, it verifies social cohesiveness for each generated spatial clique and finds (k,d)-MCCs from each clique and select the maximums.

However, a valid observation is that: if we check social cohesiveness right after a clique is generated, i.e., find the candidate (k,d)-MCC(s) from the found maximal clique before enumerating all the rest cliques, we can use the size of the largest candidate (k,d)-MCCs as a bound to stop generating unpromising cliques, i.e., cliques which are not possible to contain larger (k,d)-MCCs. Moreover, as the size of the candidate (k,d)-MCC(s) becomes larger, the pruning also becomes more effective.

As a result, in this section, we develop an efficient (k,d)-MCCs search algorithm. It is different from the baseline in two folds. Firstly, after a spatial clique is generated, we search for the (k,d)-MCCs in the clique-induced social graph immediately, and the bound will be updated as the largest size of (k,d)-MCCs found so far. Secondly, before generating a clique, we check whether the current clique search branch is able to generate candidate (k,d)-MCCs with sizes greater than the current bound, if not, we terminate the clique search branch.

The (k,d)-MCC search algorithm is shown in Algorithm 2. It is based on the maximal clique enumeration algorithm [40] which will be briefly reviewed in Section 3.2.1 with four non-trivial modifications: (1) finding candidate maximum (k,d)-MCCs immediately after generating a maximal clique (line 7); (2) terminating a search branch if no larger (k,d)-MCCs exist based on four pruning conditions (line 5), Section 3.2.2; (3) a heuristic rule to find larger (k,d)-MCCs at early stages by carefully selecting promising vertices to expand the candidates (line 10), Section 3.2.3; (4) reducing the cost of computing pruning conditions by possibly reusing previous results (related to line 5), Section 3.2.4.

Algorithm 2: effiMCCSearch(T, T')

```
1  $b \leftarrow 0$, $\mathcal{R} \leftarrow \emptyset$;
2  mccbkp($\emptyset, V(T'), \emptyset$);
3  return $\mathcal{R}$;
4  Procedure mccbkp($R, P, X$)
5      terminate this branch based on termination conditions;
6      if $P \cup X == \emptyset$ then
7          $\mathcal{R}' \leftarrow$ find maximum connected k-truss in $T(R)$;
8          $b \leftarrow$kdmccCollect($\mathcal{R}, \mathcal{R}', b$);
9      $u \leftarrow$ select a pivot from $P$ ;
10     for each $v \in P \setminus N(u, T')$ do
11         mccbkp($R \cup \{v\}, P \cap N(v, T'), X \cap N(v, T')$);
12         $P \leftarrow P \setminus \{v\}$;
13         $X \leftarrow X \cup \{v\}$;
```

3.2.1 *Revisit of Maximal Clique Enumeration*

Maximal clique enumeration. bkp [40] works on three vertex sets R, P and X and finds all the maximal cliques in T'. In each recursion state, R records the clique found so far, P contains the vertices to be added into R and X contains the vertices that were previously added into R and now have been explicitly excluded. P and X are disjoint and they together contain all the vertices that are adjacent to all the vertices in R. Initially, R and X are empty and P is $V(T')$. From P, bkp picks a $v \in P$, adds v to R and removes v's non-neighbours from P and X, i.e., $P \leftarrow P \cap N(v, T')$ and $X \leftarrow X \cap N(v, T')$. Then bkp recursively calls itself and performs the same operation on the newly generated R, P and X until the set P becomes empty. It then reports a maximal clique if the current X is empty. The reason is that if $X \neq \emptyset$, it implicitly means R is not maximal because vertices in X can be added into R to form a larger clique. After finishing the recursive search branch of adding v into R, bkp restores R, removes v from P, adds v into X, and then expands R with the next vertex in P.

Pruning search branches with pivots. Given a search state R, P and X, let $u \in P$[1], the intuition is that, cliques generated by expanding R with a vertex in $P \cap N(u, T')$ can always be further expanded by adding u subsequently. Therefore, it is safe to expand R with $P \setminus N(u, T')$ only. To pursue the maximum pruning power, a vertex u maximising $|P \cap N(u, T')|$ shall be chosen, called a pivot.

Clearly, once a maximal clique is generated, we can search for (k,d)-MCCs immediately, line 7 Algorithm 2. The largest size of the candidate (k,d)-MCCs found so far will be used as a bound. In the following, we focus on how prunings, order heuristics, computation reuse are implemented, respectively.

3.2.2 *Terminating Unpromising Branches Earlier*

The idea is that we estimate the upper bound of the (k,d)-MCCs in the current search branch. If the upper bound is smaller than the found bound b, we terminate the search branch. There are four upper bounds. (1) If $|R \cup P| < b$, we can terminate the branch. This means, if the largest possible clique is already smaller than b, the possible (k,d)-MCCs contained are thus smaller than b. (2) Let $\mathcal{K}(R \cup P)$ be the maximum connected k-truss in the induced graph $T(R \cup P)$, then $|V(\mathcal{K}(R \cup P))|$ is the upper bound. This

[1]Note that u can be chosen from $P \cup X$

is without considering spatial constraints in T'. (3) The largest possible truss number within the induced subgraph $T'(R \cup P)$ is the upper bound of the maximum clique in T'. This is without considering social constraints in T. (4) Considering both (2) and (3), we can have a more tight bound. defined based on (k, k')-truss below:

Definition 5. (k, k')-***truss***. *Given T, T' and a vertex set S such that $S \subseteq V(T) \wedge S \subseteq V(T')$, if $T(S)$ is a connected k-truss in T and $T'(S)$ is a connected k'-truss in T', we say $(T(S), T'(S))$ is a (k, k')-truss. For ease of discussion, we also call S a (k, k')-truss.*

Let k'_{max} be the largest possible truss number such that a (k, k'_{max})-truss is contained in $T(R \cup P)$ and $T'(R \cup P)$, k'_{max} is a tight upper bound of the size of (k,d)-MCCs in the current recursion branch.

The above bounds are applied one after another following the discussed order. This is because their computation cost increases accordingly and we want to terminate an unpromising branch as early as possible. If a pruning with a loose bound is enough, we can avoid computing a tighter bound expensively.

3.2.3 Search Order

Given that having a larger (k,d)-MCC size will help the algorithm terminate earlier, we design a heuristic rule aiming to obtain large (k,d)-MCCs first. The rule is as follows: given a search state with R and P, when we need to select which vertex in $P \setminus N(u, T')$ should be added into R first (line 10, Algorithm 2), we choose from the vertices in the (k, k'_{max})-truss contained in $T(R \cup P)$ and $T'(R \cup P)$ in prior because adding such vertices is likely to generate larger (k,d)-MCCs. Among the vertices in the (k, k'_{max})-truss, we consider adding the vertex v with the largest $deg(v, T')$, where ties are broken arbitrarily.

3.2.4 Computation Reuse for Pruning

Finding the upper bounds in cases (2)(3)(4) in Section 3.2.2 may not be cheap, even though truss decomposition is in polynomial time [24]. However, a nice observation is that a search state (R,P,X) and its child state (R_c,P_c,X_c) are likely to have similar truss results. Suppose $R_c = R \cup \{v\}$, $P_c = P \cap N(v, T')$, it is easy to see $R_c \cup P_c \subseteq R \cup P$. As a result, the maximum k-trusses in $T(R_c \cup P_c)$ are subsets of the maximum k-trusses in $T(R \cup P)$, hence the computation can be done incrementally using truss maintenance techniques [24] by passing the existing $T(R \cup P)$ and $T'(R \cup P)$ and truss indices to its child recursions. Similarly, (k, k'_{max})-truss can be computed incrementally as well.

On the other hand, there are some special cases where we can cheaply determine that the child state cannot be pruned: (1) if $|R \cup P| = |R_c \cup P_c|$, the child state's upper bounds are the same as the parent's; (2) let $\mathcal{K}$ be the maximum k-truss in $T(R \cup P)$, if $V(\mathcal{K}) \subseteq R_c \cup P_c$, the child state cannot be pruned; (3) let S be the (k, k'_{max})-truss in $T(R \cup P)$ and $T'(R \cup P)$, if $S \subseteq R_c \cup P_c$, the child state cannot be pruned. Proofs are omitted as correctnesses are obvious.

3.2.5 Example and Discussion

Example. We show an example using Algorithm 2 to search for (k,d)-MCCs. Given the T and T' in Figures 1 (a) and (c). Initially, $R = \emptyset$ and $P = \{1, a, \ldots, u\}$ Algorithm 2 tries to terminate the recursions by computing all four upper bound sizes, firstly producing a (4,6)-truss with vertices $\{d, e, f, g, h, i\}$. Then pivot h is selected, making Algorithm 2 only need to expand R from $P \setminus N(h, T') = \{h, r, s, q, p, o, m, n, k, \ l, t, u\}$ rather than P. Next, based on the order heuristic rule, h is added into R, reducing P to $\{a, b, c, d, e, f, g, i, j\}$. Such recursions continue until the first (k,d)-MCC, $\{d, e, h, g, f, i\}$ is discovered while bound computation can be reused from when d is added into $R = \{h\}$. After the first result is produced, b is updated to 6. Using this bound, when Algorithm 2 backtracks to the recursion state, in which r is added into R with $R = \{r\}$, $P = \{s, p, q\}$ and $X = \emptyset$. The first upper bound pruning condition is applied to terminate this branch because $|R \cup P| < 6$. Other search branches that will find cliques in Table 2 will also be pruned with the proposed termination conditions.

Discussion. In [45], Zhang et al. proposed (k, r)-core which uses k-core instead of k-truss to represent social cohesiveness. For the purpose of comparison, we adapt the AdvMax algorithm proposed in [45] for finding (k,d)-MCCs and denote it as KRM. KRM may have smaller search space, because social constraint on T is also checked during the clique enumeration. However, after incorporating social constraint check along the way, the powerful pivot-based pruning for clique enumeration cannot be used because the classic pivot pruning works only for the structure part (That might be why the work [45] used binary search rather than bkp). On the other hand, we have studied adapting the pivot idea considering both structural and social constraints. Unfortunately to determine such pivots is very complicated and the pruning power of such pivots cannot be guaranteed. Experimental performance comparison between our algorithms and KRM can be found in Section 5.

3.3 Prunings before (k,d)-MCCs Enumeration

In practice, a maximal connected k-truss T and its corresponding spatial neighbourhood network T' can be pruned before (k,d)-MCCs enumeration. The aim is to reduce the size of the input as much as possible so that (k,d)-MCC enumeration can be more efficient.

Pruning vertices in T' (I). We introduce a k-truss property first, followed by the explanation and the pruning rule.

Property 1. *For every vertex v in a k-truss graph T, v has $deg(v, T) \geq k - 1$ [12].*

Intuitively, if $T(V(T'))$ contains a k-truss, then each $v \in V(T')$ should have as least $k - 1$ neighbours in T. Accordingly, v should also have at least $k - 1$ neighbours in T'.

Pruning Rule 1. *For each $v \in V(T')$, if $deg(v, T') < k-1$, v can be pruned from T'.*

Pruning edges in T' (II). Next, we show another k-truss property which can be used to prune edges in spatial network T'. The idea is that, if two vertices are far from each other in T making them not able to be in the same k-truss, even though they are spatially close in T', their link in T' can be discarded when enumerating (k,d)-MCCs.

Property 2. *The structural diameter of a connected k-truss T with $|V(T)|$ vertices is no more than $\lfloor \frac{2|V(T)|-2}{k} \rfloor$[12].*

Pruning Rule 2. *Given T and T', let T'_0 be a connected component of T'. An edge $e(u, v) \in E(T'_0)$ can be pruned if*

$gd(u, v, T(V(T_0'))) > \lfloor \frac{2|V(T_0')|-2}{k} \rfloor$, where $gd(u, v, T(V(T_0')))$ denotes the distance between u and v in $T(V(T_0'))$.

Pruning Rule 2 is correct: suppose vertices u, v can co-exist in a connected k-truss $\mathcal{K} \subseteq T(V(T_0'))$, then we will have $gd(u, v, T(V(T_0'))) \leq gd(u, v, \mathcal{K}) \leq \lfloor \frac{2|\mathcal{K}|-2}{k} \rfloor \leq \lfloor \frac{2|V(T_0')|-2}{k} \rfloor$, this contradicts with the pruning condition.

Pruning vertices in T (III). Let D be a set of vertices pruned from T', obviously they should also be removed from T. After removing D from T, another set of vertices D' in T may be further removed due to truss maintenance [24]. D' will need to be pruned from T'.

Cascading pruning effect. We summarise the cascading pruning effect here: (1) Pruning I will cause Pruning II and III, because after pruning vertices in T', $gd(u, v, T(V(T_0')))$ becomes larger and $\lfloor \frac{2|V(T_0')|-2}{k} \rfloor$ becomes smaller, so more edges may be further pruned from T'. Also, the pruned vertices of T' should be removed from T; (2) Pruning II will cause Pruning I, because, after some edges are pruned, some vertex degrees will decrease which may lead to new vertices be pruned from T'; (3) Pruning III will cause Pruning I, which has been explained.

In the implementation, vertex pruning I, III are prioritised as they are cheap. Pruning stops if no changes are caused.

4. FINDING SPATIAL APPROXIMATE RESULT

Since all the proposed exact algorithms in Section 3 can have exponential time in the worst case, we aim to design a polynomial algorithm by relaxing spatial constraints. The polynomial algorithm can approximately find co-located communities (which are still k-trusses but have vertices within longer distances). The spatial distances can be theoretically bounded. We firstly discuss how (k,d)-MCCs should be approximated. Then we propose three types of search bound regions: upper bound region, tight bound region and error-bounded region. Using the bound regions, we design an algorithm that can find approximate results meeting a user-specified spatial error ratio requirement.

4.1 How to Approximate (k,d)-MCCs

It is desirable to have efficient algorithms to find approximate results. Accordingly, several questions are interesting: How approximation should be defined? What are good approximation results? Can users specify their own approximation preference, i.e. to what extent the discovered results are approximate? We will answer these questions.

Define approximation. Firstly, a (k,d)-MCC is considered cohesive both structurally and spatially. In general, both structural and spatial constraints can be relaxed, however, since the exponential number of exact (k,d)-MCCs comes from checking spatial constraints, we only study the approximate results with spatial constraints relaxed. Let us define an α-approximation of a (k,d)-MCC below:

Definition 6. Approximate (k,d)-MCC. *Let J be a (k,d)-MCC, J' be a (k,d')-CC satisfying $J \subseteq J'$ and $d \leq d'$, we consider J' as an α-approximation of J with spatial error ratio $\alpha = \frac{d'}{d}$, where $\alpha \in [1, +\infty)$.*

Here, J' is a k-truss with the maximum distance between vertices in $V(J')$ no more than d'. Technically, α can be less than 1, but this is not desired.

Reasonable approximation. From the definition, α-approximation of a (k,d)-MCC is not unique: the maximal α-approximation of a (k,d)-MCC is a (k,αd)-MCC, while the minimal α-approximation of a (k,d)-MCC is the (k,d)-MCC itself. Both the maximal and minimal α-approximations lead to the exponential number of co-located communities. As a result, a polynomial algorithm that can discover any α-approximations should suffice. However, superiority does exist among approximations, eg., let J_1', J_2' be two α-approximations of a particular (k,d)-MCC, if $J_1' \subseteq J_2'$, J_1' is considered better than J_2', because J_1' is "cleaner".

Specify error ratio. Ideally, users should be able to specify their preferred spatial error ratios, because different users may have different requirements. With a given spatial error ratio α, the approximate algorithm finds α-approximation results accordingly. In the next section, we introduce using spatial index to guarantee error ratio.

4.2 Spatial Index and Search Bounds

The idea of searching in polynomial is to delegate a spatial index to check spatial constraints. The outcome is, with the index, we are able to cheaply locate a region or a (limited) number of regions where we can focus on checking structural constraints only, because the user-specified spatial error ratio can be guaranteed on the k-truss results discovered within the located regions. In the following, we will introduce the spatial index first, and then elaborate on how two typical bound regions are identified and how error-bounded search regions can be identified.

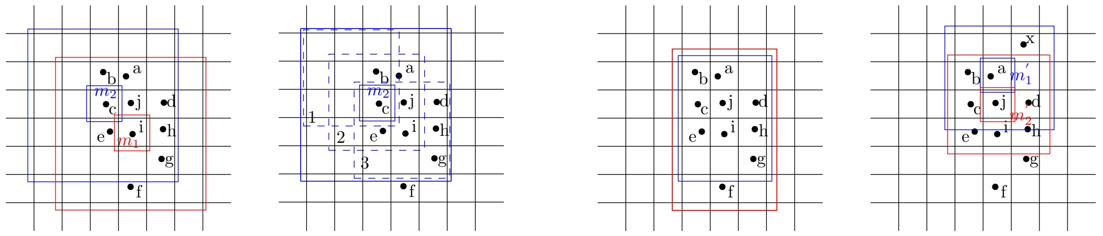
Space division. We consider the space is divided into equal-sized cells. Each cell is a $w \times w$ square. w is fixed once the space is divided. Vertices are distributed into the cells. If a cell is not empty, we call it a *landmark cell*. Rectangle region and square region are defined below, used later.

Definition 7. Rectangle region. *A rectangle region is a subspace of the entire space, with a rectangle shape containing only complete cells. Square region is defined similarly.*

Now the problem is, for each landmark cell m, to identify proper square bound regions from which k-trusses should be discovered. Two types of bound regions are interesting: (a) the upper bound region is a big bound region that can cover all the exact (k,d)-MCCs; (b) the tight bound regions are a set of regions, each of which covers some exact (k,d)-MCCs and they together cover all the exact (k,d)-MCCs. The tight bound regions can provide the best possible error ratio among all the bound regions covering the exact (k,d)-MCCs. In the following, we introduce them in detail.

Upper bound region. Given a landmark cell m and a distance d, the upper bound region r_m identified by m is an area covering all possible vertices whose distances to every vertex in m are no greater than d. Apparently, the theoretical upper bound region is irregular. However, for easy computation, we define square upper bound region as the minimal square region that covers the upper bound region, formally as: let ζ be an integer such that $(\zeta - 1)w < d \leq \zeta w$, **the square region centred at m with side size $(2\zeta+1)w$ is the square upper bound region**. In later discussions, upper bound region is used short for square upper bound region, denoted as r_m.

In Figure 2 (a), we show two landmark cells m_1 and m_2 and their upper bound regions in red and blue. Next, we show the spatial error ratio of the upper bound region.



(a) Upper bound (b) Tight bound (c) Vertex residence (d) Truss residence

Figure 2: Rectangular regions

Lemma 1. *The spatial error ratio of the upper bound region is $2\sqrt{2}+\epsilon$, where $\epsilon = \frac{3\sqrt{2}\cdot w}{d}$.*

Proof sketch. The upper bound region r_m is a square with side size $(2\zeta+1)w$. The longest distance d_{r_m} within r_m is bounded by the diagonal distance $\sqrt{2}(2\zeta+1)w$. Combining $d_{r_m} \leq 2\sqrt{2}(\zeta+1)w$ with $(\zeta-1)w < d$. We have $\frac{d_{r_m}}{d} \leq 2\sqrt{2}+\epsilon$, where $\epsilon = \frac{3\sqrt{2}\cdot w}{d}$. □

The $2\sqrt{2}+\epsilon$ error ratio may be loose in most applications, because spatial closeness has been relaxed to nearly $3d$ ($2\sqrt{2}+\epsilon \approx 3$). In the following, we introduce tight bound regions which can bound approximate results within $\sqrt{2}+\epsilon'$ error ratio.

Tight bound region. The upper bound region has side size $(2\zeta+1)w > 2d$. On the other hand, we observed that an exact (k,d)-MCC must be able to fit into a d-square (a square with side size d). This motivates us to search for the approximate results from square regions as small as possible while still not losing any exact (k,d)-MCCs. To this end, we define tight bound region formally as follows: let ζ be an integer such that $(\zeta-1)w < d \leq \zeta w$, **the square region containing m with side size $(\zeta+1)w$ is a square tight bound region.** Again, we use tight bound region short for square tight bound region. Note that, a landmark cell m can identify ζ^2 number of tight bound regions because there are ζ^2 $(\zeta+1)w$-sized squares within a $(2\zeta+1)w$-sized square.

Lemma 2. *The spatial error ratio of a tight bound region is $\sqrt{2}+\epsilon'$, where $\epsilon' = \frac{2\sqrt{2}w}{d}$.*

The proof is similar to that of Lemma 1.

In Figure 2 (b), we show three possible tight bound regions (dashed squares) given the landmark cell m_2, supposing ζ has been identified as 2. Next, we give the spatial error ratio of a tight bound region below.

The previously discussed bound regions are typical cases providing dedicated spatial error ratios. Next, we discuss how to identify bound regions satisfying a user-specified error ratio.

Error-bounded region. Given a landmark cell m, a distance d, let α be a user-given error ratio, then αd is the maximum spatial distance allowed. Let ζ' be an integer, **an error-bounded region is a square region containing the landmark cell m with side size $(\zeta'+1)w$ satisfying** $\text{argmax}_{\zeta'}\{\sqrt{2}(\zeta'+1)w \leq \alpha d | \zeta' \in \mathbb{Z}\}$. With w, d, α given, ζ' can be determined as $\lfloor \frac{\alpha d}{\sqrt{2}w} \rfloor - 1$. After that, all the square regions containing m with side size $(\zeta'+1)w$, from the square region centred at m with side size $(2\zeta'+1)w$, are retrieved as α-error-bounded regions.

4.3 Prunings

Inspecting error-bounded regions for all landmark cells is costly. If we can somehow know that the k-trusses covered by one error-bounded region r_1 is a subset of those covered by another error-bounded region r_2, the error-bounded region r_1 can be discarded. Next, we introduce how prunings can be supported by checking containment between vertex residence regions and truss residence regions.

Definition 8. Vertex residence region. *Given a square region r, the vertex residence region $Re_v(r)$ is the minimum rectangle subspace of r containing all the vertices in r.*

For example, in Figure 2 (c) shows the vertex residence regions of the two upper bound regions from Figure 2 (a) (assuming all the vertices are in the same k-truss).

Pruning Rule 3. Pruning vertex residence region. *Let r_1, r_2 be two bound regions, $Re_v(r_1)$, $Re_v(r_2)$ be two vertex residence regions, r_1 can be pruned if $Re_v(r_1) \subseteq Re_v(r_2)$.*

For example, in Figure 2 (d), since the vertex residence region within the red rectangle contains the vertex residence region within the blue, the inner blue rectangle can be pruned.

Defining vertex residence region as rectangle region is for easy computation. An alternative way is to consider only those non-empty cells as vertex residence region, however, although this approach provides better pruning power, checking containment over irregular regions is more expensive.

Similarly, a more powerful pruning condition is to consider only those k-truss vertices.

Definition 9. Truss residence region. *Given a region r, let $\mathcal{K}(r)$[2] be the k-truss contained in r, the truss residence region $Re_t(r)$ is the minimum rectangle subspace of r that contains all the vertices $V(\mathcal{K}(r))$.*

Pruning Rule 4. Pruning truss residence region. *Let r_1, r_2 be two bound regions, $Re_t(r_1)$, $Re_t(r_2)$ be two truss residence regions, if $Re_t(r_1) \subseteq Re_t(r_2)$, r_1 can be pruned.*

4.4 Error-bounded Approximation Algorithm

In this section, we introduce the error-bounded approximation algorithm. We start with a maximal connected k-truss T from the structural perspective. Recall that T does not fully satisfy the spatial constraint, i.e. every two vertices are within d. The key idea of the approximate (k,d)-MCCs search is as follows: Firstly, retrieve the landmark

[2] $\mathcal{K}(r)$ may be a disconnected graph. It may have a set of connected components, each of which is a connected k-truss.

Algorithm 3: apxSearch(T, M, α)

```
1  $b \leftarrow 0$, $\mathcal{R} \leftarrow \emptyset$;
2  prune each landmark cell $m \in M$ by applying Pruning Rules 3
   and 4 on the upper bound region $r_m$ of $m$ ;
3  sort all $m \in M$ according to the size of $\mathcal{K}(r_m)$ in descending
   order;
4  for $m \in M$ do
5      if $|V(\mathcal{K}(r_m))| \geq b$ then
6          $P \leftarrow$ generate error-bounded regions with
           $\zeta' = \lfloor \frac{\alpha d}{\sqrt{2}w} \rfloor - 1$;
7          pruning $P$ based on Pruning Rule 3;
8          sort all $p \in P$ according to $|V_p|$ in descending order;
9          for $p \in P$ do
10             if $|V_p| \geq b$ then
11                 $\mathcal{R}' \leftarrow$ maximum connected $k$-trusses in $T(V_p)$;
12                 $b \leftarrow$ kdmccCollect($\mathcal{R}', \mathcal{R}, b$);
13 return $\mathcal{R}$;
```

cells (denoted as M) which together hold all the vertices of T, $V(T) \subseteq V(M)$. Secondly, for each landmark cell $m \in M$, compute the α-error-bounded regions of m. Thirdly, for each α-error-bounded region, local maximum structural k-trusses are identified. Last, the final maximums are selected among the local maximums and will be returned as the approximate (k,d)-MCCs with guaranteed spatial error ratio α.

Algorithm 3 shows how to search approximate (k,d)-MCCs given a spatial error ratio α. It firstly prunes landmark cells using PRUNING RULES 3 and 4 . Then for each m, it generates error-bounded regions and prunes them according to PRUNING RULE 3 (lines 6 and 7). For each search region, it then computes the local approximate results in $T(V_p)$. Rather than computing k-trusses from sketch, we compute them incrementally using $\mathcal{K}(r_m)$. In this way, duplicated computation can be avoided. In order to terminate the algorithm early, the size of the best approximate results found so far is used as a bound (lines 5 and 10), and search bound regions are sorted (lines 3 and 8).

Time complexity. The time complexity of Algorithm 3 is $\mathcal{O}(|V(T)| \cdot (\lfloor \frac{\alpha d}{\sqrt{2}w} \rfloor - 1)^2 \cdot |E(T)|^{\frac{3}{2}})$. That is because, the running time of Algorithm 3 is dominated by the nested loop. The outside loop is bounded by $|V(T)|$ since there are at most $|V(T)|$ number of landmark cells. The bound of inner loop is $(\lfloor \frac{\alpha d}{\sqrt{2}w} \rfloor - 1)^2$, where $(\lfloor \frac{\alpha d}{\sqrt{2}w} \rfloor - 1)^2$ corresponds to the number of error-bounded regions. The computation inside the nested loop is dominated by truss maintenance bounded by $|E(T)|^{\frac{3}{2}}$.

4.5 Truss Attributed Quadtree Index

In this section, we show how to index the divided cells associated with useful truss pre-computations using the proposed truss attributed quad tree, denoted by TQ-tree. It can speedup the operation that retrieves vertices of a k-truss a region contains. Besides, we can have multiple choices of cell size helping locate the error-bounded regions efficiently.

TQ-tree components. Major components of the index are as below. A *truss list* contains all connected trusses in a graph G. For each truss t, an identifier is assigned, denoted as $t.id$. Since a truss with a small k may contain trusses with larger k's, the id we assign to a truss is similar to Dewey Decimal, which explicitly expresses the containment relationships. Given a vertex v, *vertex-to-truss description* returns all identifiers of the connected trusses containing v. Using Hash table, given a vertex v and a k-truss t, we can check whether t contains v in constant time. Given a truss identifier t_{id}, and a non-leaf quad q of TQ-tree, *truss-to-quad description* for q returns direct children of q that contains at least one vertex of t_{id} identified truss. Given a truss identifier t_{id}, and a leaf quad q_l of TQ-tree, *truss-vertex-to-leaf-quad description* returns all vertices of t_{id} identified truss located in q_l.

TQ-tree. A TQ-tree is a quad tree indexing all divided cells, in which the divided cells are leaf quads of TQ-tree. For each none-leaf quad in TQ-tree, it contains spatial quaternary information and is attached a list of truss-to-quad descriptions. Similarly, for each leaf quad, in addition to the bounding quad information, it also contains a list of truss-vertex-to-leaf-quad descriptions.

Retrieving vertices from a region. In this section, we show that given a T and a query region, the running time of getting vertices of T from a region r can be bounded by $\mathcal{O}(|V_r|)$ by using TQ-tree, where $|V_r|$ is the number of vertices of T in r. The idea is that: we first use the boundaries provided by r, and the description files attached in TQ-tree to explore limited branches of TQ-tree and then get the content quads via depth-first traversing. More specifically, we start from the root of TQ-tree, and collect all quads such that (1) they contain vertices in T, (2) they are in the r covering spatial space. Given a quad q in TQ-tree, since the operations that check whether a child of q is in the boundaries of r and whether the child contains T can be done in constant time, getting a vertex of T in r only depends on the height of the quad, which is determined once TQ-tree is created. Therefore, it is clear that the running time of retrieving content quads is proportional to $|V_r|$.

Searching granularity. Intuitively, different layers of TQ-tree divide the whole space into cells with different sizes. When the height of TQ-tree increases, the space is divided into cells with smaller sizes. Therefore, we may have different the search granularity. The possible number of search granularities is equal to h. The possible sizes of the search granularity is $\{2^x \times q.w | 0 \leq x \leq h\}$, where $q.w$ is the size of the leaf quads in TQ-tree.

Selecting the search granularity affects the search efficiency and effectiveness. If we select search granularity as small as a dozen meters while the query distance is over thousand of meters, constructing an error-bounded region with such small leaf quads would be low efficient. Instead, selecting a search granularity much larger than the query distance would be very fast to get the results but the error ratio would be large. We will discuss the choices of search granularity that can balance search efficiency and effectiveness in experimental studies.

5. EXPERIMENTAL RESULTS

In this section, we test all algorithms in Table 4 on a Mac with Intel i7-4870HQ (3.7GHz) and 32GB main memory.

Datasets. We conducted the experiments over five real social network datasets including Gowalla, Brightkite, Foursquare, Weibo and Twitter. Each social user contains some check-in locations. Table 5 presents the statistics for all datasets. Since we only need one check-in for each vertex, we select the most frequent check-in as the spatial coordinate for a vertex, if it has multiple check-ins.

Parameter settings. The experiments are evaluated using different settings of query parameters: k (the minimum truss

Table 4: Implemented algorithms

Name	Algorithm
Exact	Algorithm 1 + pre-pruning
EffiExact	Algorithm 2 + pre-pruning
AdvMax	Algorithm in [45] for searching a maximum (k,r)-core
AdvMaxAll	Adapted AdvMax to find all maximum (k,r)-cores
KRM	Adapted AdvMax to find all (k,d)-MCCs + pre-pruning
Apx1	Algorithm 3 with $\alpha = 2\sqrt{2} + \epsilon$ and index
Apx2	Algorithm 3 with $\sqrt{2} + \epsilon'$ and index
Apx1Ini	Apx1 without index
Apx2Ini	Apx2 without index
SAC	Algorithm Exact+ in [17]
GeoModu	Algorithm in [6]

Table 5: Statistic information in datasets

Dataset	#vertices	#edges	#checkins	k_{max}
Gowalla [10]	196,591	950,327	6,442,890	29
Brightkite [10]	58,228	214,078	4,491,143	43
Foursquare [37, 27]	4,899,219	28,484,755	1,021,970	16
Weibo [28]	1,019,055	32,981,833	32,981,833	11
Twitter [28]	554,372	2,402,720	554,372	16

number) and d (the distance threshold, in meters) as well as different settings of dataset parameters: $q.w$ (the search granularity) and n (the percentage of vertices). The ranges of parameters and their default values are shown in Table 6, in which we select reasonable k based on datasets. Furthermore, when we vary the value of a parameter for evaluation, all the other parameters are set as their default values.

Index construction. Space division: the width of minimum cells in TQ-tree for the whole space is set to be 100 meters. The index construction time and size for each dataset are displayed in Table 7.

5.1 Efficiency Evaluation

Scalability. To verify the scalability of our algorithms, AdvMaxAll (adapted AdvMax [45] for searching all maximum (k', r)-cores) and KRM (searching all (k,d)-MCCs), we choose different sizes of sub-datasets by selecting different percentages of vertices in each dataset. For AdvMaxAll, we set k' as k-1, where k is the default value for the corresponding dataset. We implemented KRM by adapting the problem setting of AdvMax from k-core to k-truss. From the results in Figures 3 (a) to (e), we can see that the exact algorithms run much slower when the data size is equal to or larger than 80%. However, for the approximate algorithms, their time costs increase almost linearly as the data sizes increase for all datasets. For all datasets, our EffiExact outperforms KRM 30% and AdvMaxAll outperforms EffiExact 10% in average. Surprisingly AdvMaxAll does not outperform EffiExact significantly. This is mainly because the size of candidate vertices of searching (k,r)-cores is larger than that of searching (k,d)-MCCs on all real datasets. In the following experiments, we focus only on our algorithms(i.e., excluding KRM and AdvMaxAll).

Effect of k. Figures 3 (f) to (j) evaluate the performance of the algorithms when we vary the value of k. In general, all algorithms take less time when k increases. This is because increasing k will result in the decrease of the sizes of k-trusses. The EffiExact runs consistently faster than Exact, especially when k is large. In addition, the study shows that the approximate algorithms outperform the exact algorithms greatly. The performance can be improved in two orders of magnitude in average over all datasets. Compared

Table 6: Parameter settings

Parameter	Range	Default value
k	Gowalla, Brightkite: 5,7,9,11,13,15,17	11
	Weibo: 5,6,7,8,9,10,11	9
	Twitter Foursquare:3,5,7,9,11,13,15	11
d	$500, 1000, 1500, 2000, 2500, 3000$	2000
$q.w$	$100, 200, 400, 800, 1600$	400
n	$20\%, 40\%, 60\%, 80\%, 100\%$	100%

Table 7: TQ-tree construction

Dataset	Time (Sec)	Space (MB)
Gowalla	101	31
Brightkite	75	17
Foursquare	5102	1680
Weibo	4812	1423
Twitter	473	152

with Apx2, Apx1 is much faster due to its low accuracy guarantee. Although Apx2 is slower, it can provide more effective results, which will be discussed in Section 5.2.

Effect of d. Figures 3 (k) to (o) show the time cost when we vary the distance value of d from 500 to 3000. With the increase of d, the exact algorithms consume time exponentially. This is because increasing d will require the algorithms to explore larger spatial neighbourhood network. The experimental results also confirm our theoretical analysis that the time complexities of exact algorithms are exponential to the size of spatial neighbourhood network. In our experiment, EffiExact is faster than Exact by 3-5 times. Different from exact algorithms, the time cost of approximate algorithms increases slowly for all datasets. In most cases, the approximate algorithms can answer the (k,d)-MCC search within 10 seconds, which is able to answer the real time search. Only for Foursquare, it requires to consume about 40 seconds for answering the (k,d)-MCC search.

Effect of granularity. Figures 3 (p) to (t) demonstrate the time cost when we vary the search granularity. To show the power of index, we also implemented algorithms Apx1 and Apx2 without the support of index, denoted as Apx1Ini and Apx2Ini respectively. Overall speaking, the time cost of both algorithms decreases as the search granularity increases. This is because the space would be divided into less number of cells for the larger search granularity, and the time complexity is proportional to the number of cells.

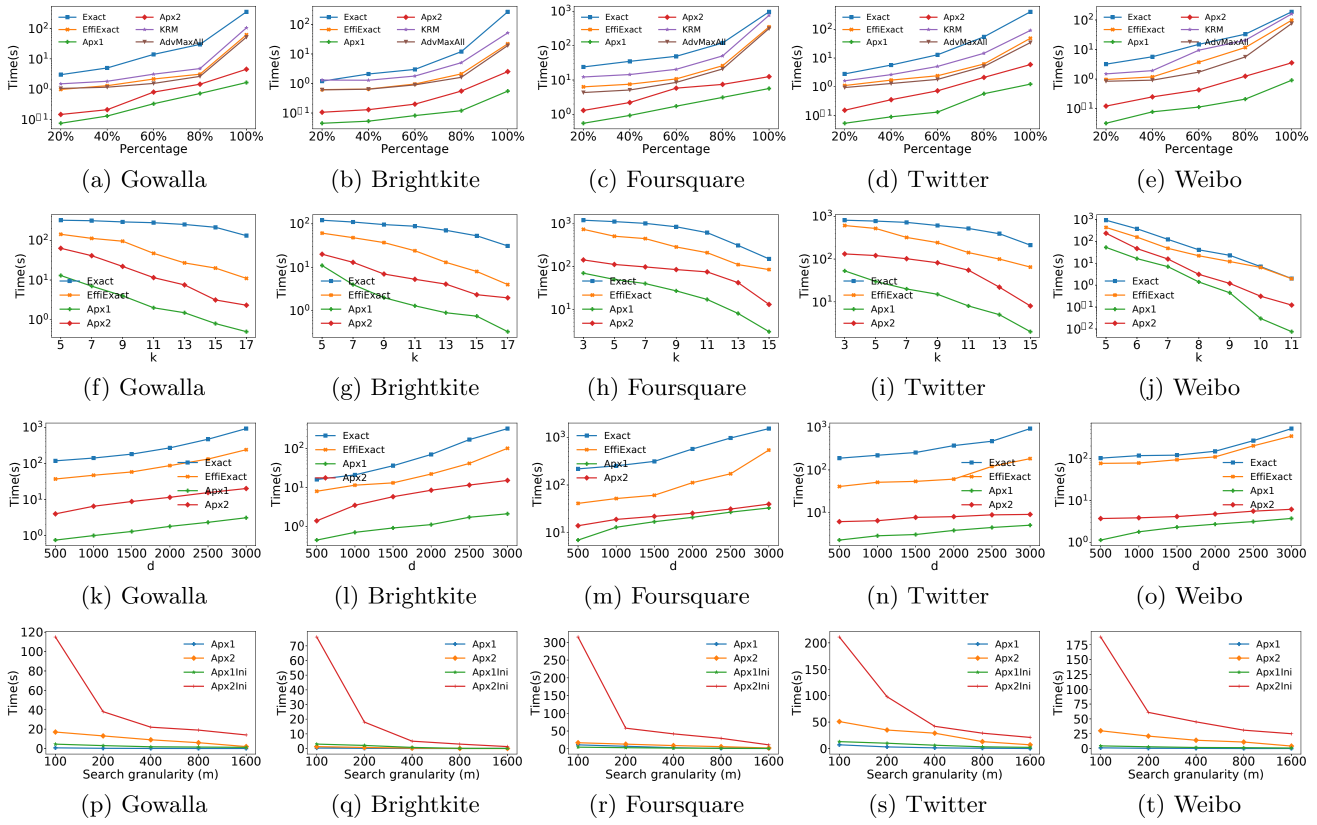
5.2 Effectiveness Evaluation

5.2.1 Exact Algorithms

We present the trends of pre-pruning effectiveness in exact algorithms when the parameters k and d vary. To show the effectiveness, let's introduce two metrics as below.

Metrics. Let T be the union of maximal connected k-trusses in G. Let G' be the graph after pruning. The vertex pruning ratio is measured by $\frac{|V(G')|}{|V(T)|}$. Let t_1 and t_2 be the running time of a algorithm applying and without applying pruning rules. The time saved ratio is defined as $\frac{t_2 - t_1}{t_2}$.

Effect of k. Figure 4 (a) reports the vertex pruning ratios as we change the k value. For datasets Gowalla and Brightkite, their pruning effectiveness becomes higher as k value increases. Interestingly, for the datasets Weibo, Foursquare and Twitter, their pruning effectiveness becomes higher at the beginning and then decreases when the value of k increases further. The main reason is that, in the three datasets, their vertices have good spatial and social dis-



(a) Gowalla (b) Brightkite (c) Foursquare (d) Twitter (e) Weibo

(f) Gowalla (g) Brightkite (h) Foursquare (i) Twitter (j) Weibo

(k) Gowalla (l) Brightkite (m) Foursquare (n) Twitter (o) Weibo

(p) Gowalla (q) Brightkite (r) Foursquare (s) Twitter (t) Weibo

Figure 3: Efficiency evaluation

tributions, i.e., the vertices with higher social cohesiveness also tend to have spatial closeness with each other. Therefore, the pruning effectiveness becomes less significant when k is high. Actually, similar trends occur in Gowalla and Brightkite if we further increase the value of k.

Effect of d. Figure 4 (b) shows the vertex pruning effectiveness ratio when we change the value of d. For all datasets, the pruning effectiveness decreases with the increase of d value. This is because as d increases, the spatial cohesiveness constraint is relaxed so that the explored spatial network becomes large, which makes less number of vertices to be filtered. However, when d is at the interval of 1500 to 2000 meters, our pruning technique can prune more than 50% vertices averagely, which makes EffiExact run much faster than Exact in all configurations.

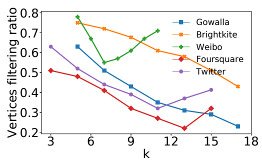
Effect of pruning rules. Figure 5 (a) reports the vertex pruning ratios with the left scale and time saved ratios with the right scale for pruning vertices (P1) and edges (P2) in T', and pruning vertices in T (Truss) individually, or applying these rules interchangeably by cascading pruning (CAS). The coloured bars correspond to pruning ratios while the bars with hatches correspond to time saved ratios when applying pruning rules in EffiExact. For all datasets, it shows that applying rules individually has limited pruning effectiveness with less than 15% vertices pruned, and less than 12% of time saved by P1(P1TS) and P2(P2TS), though a bit improved by Truss. However, applying these rules interchangeably can prune much more vertices and save much more time. For Gowalla and Weibo, over 60% of vertices can be filtered out and more than 50% of time can be saved (shown by CAS(CASTS)).

5.2.2 *Approximate Algorithms*

Region pruning ratio. Region pruning ratio is defined as the ratio of the number of regions to be evaluated overs the number of regions actually evaluated.

Figure 5 (b) shows the region pruning ratios with left scale (time saved ratio with right scale) when applying pruning rules 3 and 4. The bars P31(P31TS) and P32(P32TS) demonstrate the pruning ratios (time saved ratios) when applying pruning rule 3 in Apx1 and Apx2. In all datasets, pruning rule 3 is more effective when pruning tight bound regions compared to applying it to prune upper bound regions. However, the pruning ratio of pruning rule 4 (shown by P4) outperforms rule 3 when pruning tight bound regions for all datasets.

Figure 6 (a) shows the upper bound region pruning ratios when vary the search granularity for algorithm Apx1. Overall speaking, the pruning ratio decreases as search granularity increases for all datasets. This is because smaller search granularity makes the region become more compact. The pruning ratio for Foursquare is slightly worse than others but the average pruning ratio is still over 0.5. Figure 6 (b), shows the tight bound region pruning ratios when varying the search granularity for algorithm Apx2. The pruning effect trend is similar to upper bound region pruning.


(a) Varying k

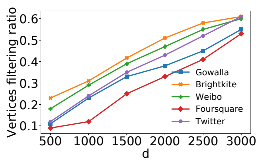
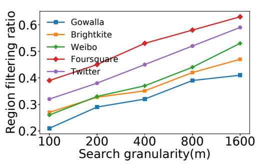

(b) Varying d

Figure 4: Exact algorithm pruning effectiveness


(a) Apx1

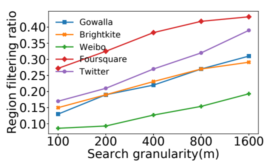
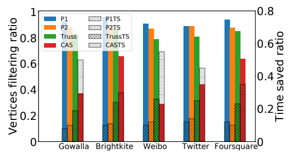

(b) Apx2

Figure 6: Region pruning ratio


(a) Pruning rules 1 and 2

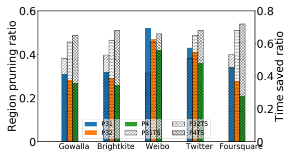
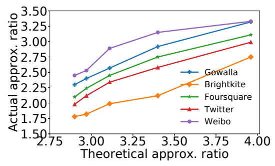

(b) Pruning rules 3 and 4

Figure 5: Effectiveness of pruning rules


(a) Apx1

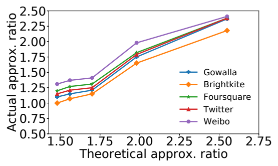

(b) Apx2

Figure 7: Approximation ratio

Effect of granularity on α. We show the correlation between the theoretical and actual approximation ratios when we run the algorithms Apx1 and Apx2 over all datasets, and show the trend when the search granularity changes. They are plotted in Figures 7 (a) and (b). The x-axis in the figures is the theoretical approximation ratios. For Apx1, the theoretical approximation ratios are 2.9, 2.97, 3.11, 3.4, and 3.96. For the Apx2, the theoretical approximation ratios are 1.48, 1.56, 1.70, 1.98, and 2.55. From the experimental results, we can see that the real approximation ratios of Apx1 and Apx2 are much smaller than theoretical approximation ratios. For both algorithms, real approximation ratios increase as the search granularity increases. The approximation ratio of Apx1 increases almost linearly as the theoretical approximation ratio increases. For Apx2, we can observe a sudden increase for all datasets after the search granularity goes beyond 400. Within 400, we can see that the actual approximation ratios are all less than 1.4. Therefore, in real applications we may need to tune the search granularity to balance the search efficiency and effectiveness.

5.2.3 *Spatial Density*

We verify the spatial closeness of (k,d)-MCCs found by our algorithms Exact, Apx1 and Apx2 by comparing with that of the state-of-the-art spatial-aware community models: SAC [17], GeoModu [6] and AdvMax [45]. The spatial closeness is evaluated by spatial density. Given a community J with spatial diameter d, the spatial density of J is defined as: $\frac{\sum_{u,v \in V(J)} ed(u,v)}{2|V(J)|}$.

SAC. It finds a k-core G', containing a query vertex and vertices are spatially contained in a minimum covering circle with smallest radius. We randomly select 200 query vertices and set up the structural cohesiveness as 12-core.

GeoModu. It refines the weight of each edge $e_{u,v}$ in graph G as $\frac{1}{d^{\eta}_{u,v}}$ where $d^{\eta}_{u,v}$ is the normalised spatial distance from u to v, where η is a decay factor and is set to 1. It then detects the communities using modularity maximisation.

AdvMax. It finds the maximum (k,r)-core which is a k-core containing pairwise vertices with spatial distance no greater than d and maximising the cardinality.

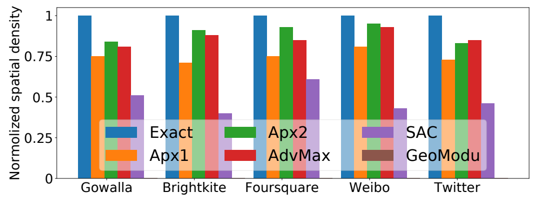
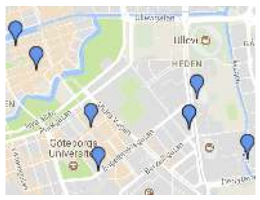


Figure 8: Density study


(a) (4,1.5km)-MCC

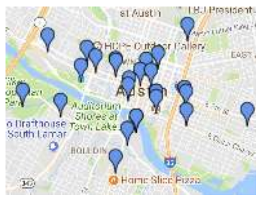

(b) (9,5km)-MCC

Figure 9: Case study

For Exact, Apx1, Apx2 and AdvMax, we setup k=11 (k=10 for AdvMax), randomly select 200 different query distances between 500 to 2000 meters, generate results, and compute average spatial density for each dataset. All results are normalised by $\frac{\mathcal{D}-\mathcal{D}_{min}}{\mathcal{D}_{max}-\mathcal{D}_{min}}$, where $\mathcal{D}$ is an average spatial density, $\mathcal{D}_{max}$ and $\mathcal{D}_{min}$ are the extremes.

Figure 8 shows that Exact performs the best and outperforms AdvMax due to its structural tightness. AdvMax also performs reasonably well because both Exact and AdvMax tend to enlarge the cardinality as much as possible for the given distance threshold. As expected, the approximate algorithms Apx1, Apx2 perform worse than Exact but reasonably acceptable. As shown in the figure, Apx2 performs better than AdvMax in most datasets except for Weibo. Also Apx2 performs better than Apx1 in all datasets due to lower error ratio. Compared with the above algorithms, SAC and GeoModu perform much worse mainly because they don't intend to include as many vertices as possible. GeoModu is listed as 0 after normalisation.

5.2.4 *Case Study*

We conducted two case studies on Gowalla to show the effectiveness of (k,d)-MCCs. In contrast to connected k-trusses, our models can ensure the spatial closeness over community members.

Figure 9 (a) shows a community with k=4 and d=1.5km. All the members are around Gothenburg University in Sweden. The community members are good candidates for meet-up activities since they have strong social relationships, i.e. each member has 3 friends in the community and members who are not friends are connected by their mutual friends; and the longest distance over them is bounded by 1.5km.

Figure 9 (b) shows a community with k=9 and d=5km. We can see that there are some members around the downtown in Austin that are very close with each other. And there are some members in outskirt that are relatively distant to members in the downtown area. Removing any of these relatively distant members makes the community collapse from social cohesiveness perspective. Although the query has d = 5km, the actual distance between most members is less than 3.3km.

6. RELATED WORK

Global community search. In general global community search methods search top-k communities matching the query parameter and having the global highest score for some goodness metrics. Li et al. [31] consider k-clique as structural cohesiveness metric and consider outer influence score as a goodness metric. In [30], k-core is used to measure the social cohesiveness and internal influential score is used to rank the communities. Qin et al. [35] find top-k most densest subgraphs from a graph using proposed local density score schema. In [45], they find (k, r)-core community such that socially the vertices in (k, r)-core is a k-core and from similarity perspective pairwise vertices similarity is more than a threshold r. Our work is different from them in twofold. From modelling perspective we use k-truss to measure social cohesiveness. The k-truss is superior over k-core since k-truss provides connectivity guarantee. From technical perspective, we focus on finding both exact and spatial approximate maximum communities.

Local community search. The goal of local community search is to search communities containing a set of query vertices. In [24, 2], maximal triangle-connected k-truss containing a query vertex metric are considered as communities. Cui et al. [14], search local optimal community modelled as connected subgraphs, containing the query vertices and maximising the minimum degree of each vertex in the subgraphs. In [18, 25, 29], they consider both the structure and keyword cohesiveness when searching the community. A k-clique based model is proposed in [44], in which a community is defined as the maximal k-clique adjacent connected subgraphs containing all query vertices, named as k-clique percolation community. Hu et.al. [23] propose algorithms searching a community that is a minimal connected Steiner tree containing all query vertices while maximising the cardinality. These works do consider spatial factor when modelling a community. Recently, Fang et al. [17] propose local community search on spatial network. Our work is different from them. We consider pairwise vertices distance as a query parameter while they focus on minimising the spatial pairwise vertices distance. They search the local optimal community containing a query vertex while we search the communities that matches the query parameters and has the maximum cardinality globally.

Table 8: Community models comparison

	GCD & LCD	GCS	LCS
Non spatial	[19, 20, 21, 34, 36, 43]	[31, 30, 35, 45]	[18, 39, 14, 13, 24, 44, 23]
Spatial	[16, 6]	(k,d)-MCCs	[17]

Community detection. Community detection methods usually detect all communities in a graph [19, 20, 21, 34, 36]. These works employ link-based analysis to detect the communities. Recently, works [16, 6] merge spatial constraints into the link-based analysis. In [16], gravity model is used when defining modularity function. In [6], a distance decay function is used when defining modularity function. Our work is very different from them. Firstly, these algorithms detect all communities once parameter is given by exploring the whole graph while our proposed methods explore data associating with query parameters only. Secondly, we have clear user specified query parameters.

Cohesive subgraph mining. Cohesive subgraph is an important concept in social network analysis. Many works are based on these cohesive subgraphs such as maximal clique [8, 9], k-core [38, 7, 32, 26], k-truss [12, 41], k-compact tree [33], DN-graph [42], and k-edge connected subgraph [46, 5, 3].

Distance bounded vertex set(s) searching. Aggarwaal et.al. [1] and Clark et.al. [11] find that finding a maximum clique in a spatial neighbourhood network is polynomial solvable. And accordingly, there is a polynomial algorithm for the problem that finding k vertices with minimum diameter [1] and the performance is further improved by [15]. Gupta et al. [22] report the total number of maximal cliques could grow exponentially with the size of a spatial network and propose algorithms that generate slightly super-maximal cliques.

Table 8 categorises the related works in terms of the type of method for finding communities with or without spatial cohesiveness requirement, and positions our (k,d)-MCCs work with the comparison to other works.

7. CONCLUSION

In this paper, we study the maximum co-located community search problem in large scale social networks. We propose a novel community model, co-located community, considering both social and spatial cohesiveness. We develop efficient exact algorithms to find all maximum co-located communities. We design approximation algorithm with guaranteed spatial error ratios. We further improve the performance using proposed TQ-tree index. We conduct extensive experiments on large real-world networks, and the results demonstrate the effectiveness and efficiency of the proposed algorithms.

8. ACKNOWLEDGMENTS

This work is jointly supported by the ARC Discovery Projects under Grant No. DP170104747, DP160102412, DP160102114 and DP180100212, the NSF of China under Grant No. U1736104 and 61532021. We would like to thank anonymous reviewers for their helpful comments.

9. REFERENCES

[1] A. Aggarwal, H. Imai, N. Katoh, and S. Suri. Finding k points with minimum diameter and related problems. *Journal of Algorithms*, 12(1):38 – 56, 1991.

[2] E. Akbas and P. Zhao. Truss-based community search: a truss-equivalence based indexing approach. *PVLDB*, 10(11):1298–1309, 2017.

[3] T. Akiba, Y. Iwata, and Y. Yoshida. Linear-time enumeration of maximal k-edge-connected subgraphs in large networks by random contraction. In *CIKM*, pages 909–918, 2013.

[4] G.-R. Cai and Y.-G. Sun. The minimum augmentation of any graph to a k-edge-connected graph. *Networks*, pages 151–172, 1989.

[5] L. Chang, J. X. Yu, L. Qin, X. Lin, C. Liu, and W. Liang. Efficiently computing k-edge connected components via graph decomposition. In *SIGMOD*, pages 205–216, 2013.

[6] Y. Chen, J. Xu, and M. Xu. Finding community structure in spatially constrained complex networks. *International Journal of Geographical Information Science*, 29(6):889–911, 2015.

[7] J. Cheng, Y. Ke, S. Chu, and M. T. zsu. Efficient core decomposition in massive networks. In *ICDE*, pages 51–62, April 2011.

[8] J. Cheng, Y. Ke, A. W.-C. Fu, J. X. Yu, and L. Zhu. Finding maximal cliques in massive networks. *ACM Trans. Database Syst.*, 36(4):21:1–21:34, Dec. 2011.

[9] J. Cheng, L. Zhu, Y. Ke, and S. Chu. Fast algorithms for maximal clique enumeration with limited memory. In *SIGKDD*, pages 1240–1248, 2012.

[10] E. Cho, S. A. Myers, and J. Leskovec. Friendship and mobility: User movement in location-based social networks. In *SIGKDD*, pages 1082–1090, New York, NY, USA, 2011. ACM.

[11] B. N. Clark, C. J. Colbourn, and D. S. Johnson. Unit disk graphs. *Discrete Mathematics*, 86(1):165 – 177, 1990.

[12] J. Cohen. Trusses: Cohesive subgraphs for social network analysis. *National Security Agency Technical Report*, 16, 2008.

[13] W. Cui, Y. Xiao, H. Wang, Y. Lu, and W. Wang. Online search of overlapping communities. In *SIGMOD*, pages 277–288, 2013.

[14] W. Cui, Y. Xiao, H. Wang, and W. Wang. Local search of communities in large graphs. In *SIGMOD*, pages 991–1002, 2014.

[15] D. Eppstein and J. Erickson. Iterated nearest neighbors and finding minimal polytopes. *Discrete & Computational Geometry*, 11(3):321–350, Mar 1994.

[16] P. Expert, T. S. Evans, V. D. Blondel, and R. Lambiotte. Uncovering space-independent communities in spatial networks. *Proceedings of the National Academy of Sciences*, 108(19):7663–7668, 2011.

[17] Y. Fang, R. Cheng, X. Li, S. Luo, and J. Hu. Effective community search over large spatial graphs. *PVLDB*, 10(6):709–720, 2017.

[18] Y. Fang, R. Cheng, S. Luo, and J. Hu. Effective community search for large attributed graphs. *PVLDB*, 9(12):1233–1244, 2016.

[19] S. Fortunato. Community detection in graphs. *Physics reports*, 486(3):75–174, 2010.

[20] M. Girvan and M. E. Newman. Community structure in social and biological networks. *Proc. Natl. Acad. Sci. USA*, 99(cond-mat/0112110):8271–8276, 2001.

[21] D. Guo. Regionalization with dynamically constrained agglomerative clustering and partitioning (redcap). *International Journal of Geographical Information Science*, 22(7):801–823, 2008.

[22] R. Gupta, J. Walrand, and O. Goldschmidt. Maximal cliques in unit disk graphs: Polynomial approximation. In *INOC*, 2005.

[23] J. Hu, X. Wu, R. Cheng, S. Luo, and Y. Fang. Querying minimal steiner maximum-connected subgraphs in large graphs. In *CIKM*, pages 1241–1250, New York, NY, USA, 2016. ACM.

[24] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu. Querying k-truss community in large and dynamic graphs. In *SIGMOD*, pages 1311–1322, 2014.

[25] X. Huang and L. V. S. Lakshmanan. Attribute-driven community search. *PVLDB*, 10(9):949–960, 2017.

[26] W. Khaouid, M. Barsky, V. Srinivasan, and A. Thomo. K-core decomposition of large networks on a single pc. *PVLDB*, 9(1):13–23, 2015.

[27] J. J. Levandoski, M. Sarwat, A. Eldawy, and M. F. Mokbel. Lars: A location-aware recommender system. In *ICDE*, pages 450–461, April 2012.

[28] G. Li, S. Chen, J. Feng, K.-l. Tan, and W.-s. Li. Efficient location-aware influence maximization. In *SIGMOD*, pages 87–98, New York, NY, USA, 2014. ACM.

[29] J. Li, C. Liu, and M. S. Islam. Keyword-based correlated network computation over large social media. In *ICDE*, pages 268–279. IEEE, 2014.

[30] J. Li, X. Wang, K. Deng, X. Yang, T. Sellis, and J. X. Yu. Most influential community search over large social networks. In *ICDE*, pages 871–882, April 2017.

[31] R.-H. Li, L. Qin, J. X. Yu, and R. Mao. Influential community search in large networks. *PVLDB*, 8(5):509–520, 2015.

[32] R. H. Li, J. X. Yu, and R. Mao. Efficient core maintenance in large dynamic graphs. *TKDE*, 26(10):2453–2465, Oct 2014.

[33] C. Liu, L. Yao, J. Li, R. Zhou, and Z. He. Finding smallest k-compact tree set for keyword queries on graphs using mapreduce. *WWWJ*, 19(3):499–518, May 2016.

[34] M. E. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical review E*, 69(2):026113, 2004.

[35] L. Qin, R.-H. Li, L. Chang, and C. Zhang. Locally densest subgraph discovery. In *SIGKDD*, pages 965–974, 2015.

[36] M. Rezvani, W. Liang, C. Liu, and J. X. Yu. Efficient detection of overlapping communities using asymmetric triangle cuts. *TKDE*, pages 1–1, 2018.

[37] M. Sarwat, J. J. Levandoski, A. Eldawy, and M. F. Mokbel. An efficient and scalable location-aware recommender system. *TKDE*, 26(6):1384–1399, June 2014.

[38] S. B. Seidman. Network structure and minimum degree. *Social Networks*, 5(3):269 – 287, 1983.

[39] M. Sozio and A. Gionis. The community-search problem and how to plan a successful cocktail party. In *SIGKDD*, pages 939–948, 2010.

[40] E. Tomita, A. Tanaka, and H. Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theoretical Computer Science*, 363(1):28–42, 2006.

[41] J. Wang and J. Cheng. Truss decomposition in massive networks. *PVLDB*, 5(9):812–823, 2012.

[42] N. Wang, J. Zhang, K.-L. Tan, and A. K. H. Tung. On triangulation-based dense neighborhood graph discovery. *PVLDB*, 4(2):58–68, 2010.

[43] Y. Wu, R. Jin, J. Li, and X. Zhang. Robust local community detection: On free rider effect and its elimination. *PVLDB*, 8(7):798–809, 2015.

[44] L. Yuan, L. Qin, W. Zhang, L. Chang, and J. Yang. Index-based densest clique percolation community search in networks. *TKDE*, pages 1–1, 2018.

[45] F. Zhang, Y. Zhang, L. Qin, W. Zhang, and X. Lin. When engagement meets similarity: efficient (k, r)-core computation on social networks. *PVLDB*, 10(10):998–1009, 2017.

[46] R. Zhou, C. Liu, J. X. Yu, W. Liang, B. Chen, and J. Li. Finding maximal k-edge-connected subgraphs from a large graph. In *EDBT*, pages 480–491, 2012.