Finding Effective Geo-social Group for Impromptu Activities with Diverse Demands

Lu Chen[†], Chengfei Liu[†], Rui Zhou[†], Jiajie Xu[‡], Jeffrey Xu Yu[∓], Jianxin Li[§]
[†]Swinburne University of Technology, [‡]Soochow University,
[∓]The Chinese University of Hong Kong, [§]Deakin University
[†]{luchen, cliu, rzhou}@swin.edu.au, [‡]xujj@suda.edu.cn, [∓]yu@se.cuhk.edu.hk, [§]jianxin.li@deakin.edu.au

ABSTRACT

Geo-social group search aims to find a group of people proximate to a location while socially related. One of the driven applications for geo-social group search is organizing an impromptu activity. This is because the social cohesiveness of a found geo-social group ensures a good communication atmosphere for the activity and the spatial closeness of the geo-social group reduces the preparation time for the activity. Most existing works treat geo-social group search as a problem that finds a group satisfying a single social constraint while optimizing the spatial proximity. However, since different impromptu activities have diverse demands on attendees, e.g. an activity could require (or prefer) the attendees to have skills (or favorites) related to the activity, the existing works cannot find this kind of geo-social groups effectively. In this paper, we propose a novel geo-social group model, equipped with elegant keyword constraints, to fill this gap. We propose a novel search framework which first significantly narrows down the search space with theoretical guarantees and then efficiently finds the optimum result. To evaluate the effectiveness, we conduct experiments on real datasets, demonstrating the superiority of our proposed model. We conduct extensive experiments on large semi-synthetic datasets for justifying the efficiency of the proposed search algorithms.

CCS CONCEPTS

• **Mathematics of computing** → *Graph algorithms*; • **Information systems** → *Data mining*.

KEYWORDS

Geo-social Group; Truss; Attributed Graph

ACM Reference Format:
Lu Chen, Chengfei Liu, Rui Zhou, Jiajie Xu, Jeffrey Xu Yu, Jianxin Li. 2020. Finding Effective Geo-social Group for Impromptu Activities with Diverse Demands. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20), August 23–27, 2020, Virtual Event, CA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3394486.3403114

1 INTRODUCTION

As the geo-social networks become popular, finding geo-social groups has drawn great attention in recent years. In general, geo-social group search problem [2, 17–19, 25] aims to find a group that is socially cohesive while spatially closest to a location, which is different from the social-aware spatial keyword search works such as [1, 10, 15, 21] that consider various objectives together as an aggregate objective function and find the optimum result w.r.t. the objective function.

Most existing approaches for finding a geo-social group are mainly based on the nearest neighbour (NN) search framework. This framework progressively adds vertices that potentially satisfy the social constraint according to nearest neighbour order (w.r.t. the activity location), while checking the social constraint after each vertex is added. It returns the optimum result when it finds a subgraph satisfying the social constraint for the first time. This framework is efficient when considering a single social constraint that is cheap to check.

One of the most motivating applications for geo-social group search is instant formation of impromptu (pop-up) activities. This is because of two nice proprieties of geo-social groups. Firstly, the social cohesiveness of a geo-social group ensures the members are socially close within the group, which is a key to ensure a good communication atmosphere for the activity. Secondly, subjecting to the social cohesiveness, a geo-social group is the one that is closest to the location of the activity, which reduces the waiting time for the activity greatly.

However, since most of the existing geo-social group studies only focus on social constraint while optimizing the spatial closeness, they become less effective to discover participants for impromptu activities with diverse demands. Let us consider a real event happened in 2019. A small town in Australia was devastated by the severe bushfire, which results in at least 11 properties damaged and 33 people injured. The town needs community spirit to rebuild. This naturally arises the needs of several activities with diverse demands. A group needs to be formed urgently to react on the disaster, with at least 3 members having expertise in building temporary accommodations, 5 doctors, 4 psychologists, 2 members having expertise in community support, etc. Each member may contribute to as many skills as possible in this kind of group. Due to damaged properties, a construction team also needs to be built for rebuilding these properties, with members having different skills, such as at least 2 architects, 11 members handling masonry, 5 members dealing with welding, etc. Due to the intensive labouring, each member may contribute at most 2 skills in this construction team since multi-tasking may lead to multi-failing. Due to the disaster, people may suffer a lot mentally. To help relief psychological pressure from these people, it would be great to organize an improvised music show to soothe them, which needs to discover musicians to form a band. The found musicians may be able to play multiple instruments. However, since they perform as a band, each of them shall focus on a single instrument. From these examples, it is clear that, apart from social cohesiveness and spatial closeness requirements, an effective geo-social group model for impromptu activities with diverse demands should allow people to express 1) collective capabilities of the group w.r.t. a particular skill, e.g., at least 3 members have expertise in building temporary accommodation; and 2) capacity of each member on the maximum contribution the member can make, e.g., at most 2 skills in a construction team.

Our geo-social group model. As far as we know, none of the existing geo-social group models could handle all the above requirements. In this paper, we aim to propose a novel model to fill in this gap. We model a geo-social network with textual, social and spatial information as graph data, in which every vertex (user) is attached with a set of keyword attributes and a spatial location. To effectively discover a geo-social group discussed above, we allow users to provide query keywords describing the activity demands and keyword parameters expressing the requirements of collective capabilities of the group and contribution capacity of group members, apart from providing a social cohesiveness parameter setting up social cohesiveness and a query location. Our proposed geo-social model contains constraints and searching objective as follows.

Minimum keyword and capacity constraints (MKCC). A group satisfies MKCC if it satisfies both minimum keyword constraint MKC and capacity constraint simultaneously defined below. We allow users to provide integer ρ_i for every query keyword k_i. A group of members satisfy MKC if for every query keyword k_i, the number of members whose keyword attributes contribute to k_i is at least ρ_i. We also allow users to provide an integer r for expressing capacity for each member. A group satisfies the capacity constraint, if the keyword attributes of every member contribute to at most r query keywords. Without parameters ρ_i for every keyword k_i and r, the found group cannot satisfy the requirements of collective capabilities of the group and contribution capacity of group members of an impromptu activity.

Social constraint and spatial closeness objective. We adopt c-truss [6] for measuring social constraint, i.e., every pair of friends (vertices contained in an edge) in the geo-social group must have at least c-2 common friends. Compared to minimum degree constraint [25], a group satisfying trussness has a *guarantee of communication cost.* The spatial closeness between a query location λ and a group is measured by the Euclidean distance between λ and the member in the group most distant to λ. The geo-social group that is closest to the query location subjecting to both keyword and social constraints serves as our searching objective.

Since our proposed group model satisfies $\underline{\text{m}}$inimum $\underline{\text{k}}$eyword, $\underline{\text{c}}$apacity and $\underline{\text{s}}$ocial constraints while optimizing $\underline{\text{s}}$patial closeness, we name the proposed model as MKCSSG in this paper.

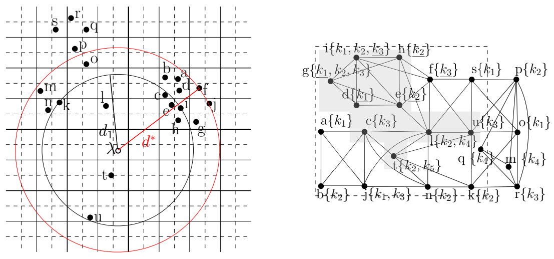
We would like to highlight that, MKCSSG is a general geo-social group model. Its instances can not only suite for the activities with diverse demands but also serve for applications of existing models. Most geo-social group models such as [25] are the instances of our model by ignoring MKCC. The SSTQ model in [19] is an instance of our model by limiting ρ_i to 1 for every query keyword k_i while ignoring the capacity constraint.

Searching Challenges. Efficiently finding MKCSSG is challenging. This is because the reasons below. First of all, checking MKCC as well as checking trussness constraint individually are non-trivial. One time trussness constraint checking costs $O(|E(G)|^{1.5})$ for graph G. MKCC checking is an open problem to the best of our knowledge and it is not cheap since it shares some similarities with set cover problem. Secondly, efficiently checking MKCC, social constraint and spatial closeness together has not been explored in the existing studies. If we use the NN based framework to solve our problem, there will be $|V(G)|$ times of MKCC and trussness constraint checking, which is extremely slow. We can optimize the NN based framework by using a R-tree embedded with social, and keyword information, which would reduce the search space in practice and have good search efficiency in some cases. However it still suffers from high time complexity. As such, three open problems are raised. Firstly, can we solve MKCC checking in polynomial time? Secondly, can we have a search framework that can quickly narrow the search space with a theoretical bound for the size of the narrowed search space while preserving the correct result? Thirdly, can the novel search framework elegantly bound the times of both MKCC and trussness constraint checking such that the times is constant regardless of $|V(G)|$?

Our approach. In this paper, we will tackle the three open problems by devising a novel algorithm for MKCC checking and proposing a novel search framework. We discover that MKCC can be solved in polynomial time by solving a min cut problem in a proposed flow network dedicated for this problem. The proposed search framework contains expanding and reducing stages. The expanding stage approaches to a search space that is sufficient large to contain the optimum result at a cost equivalent to *constant times* of MKCC and social constraint checking. The approached search space is no greater than the size of the optimum search space with a ratio of parameterized constant, which vastly restricts the search space for the reducing stage. The reducing stage progressively removes the vertex most distant to the query location in the approached search space. To reduce the time complexity of the reducing stage, we discover that the successive MKCC checking can be done incrementally, which reduces the cost of multiple times (determined by the approached search space) of MKCC checking to constant times (regardless of the approached search space) of MKCC checking. To further reduce the cost of MKCC and social constraint checking for expanding and reducing stages, we propose keyword aware union and keyword aware spanning forest data structures respectively.

Contribution. Our predominant contributions in this paper are summarised as follows.

- A novel geo-social group model: we propose a novel geo-social group model considering collective capabilities of the group and capacity of contribution from group members. (**Section** 2)
- Efficient algorithms with theoretical guarantees:
  - We devise a novel search framework and optimisation techniques, which ensure that the proposed geo-social group can be discovered with time complexity the same as the time complexity of one time MKCC checking on a small graph. (**Sections** 3, 4 and 5)



(a) Spatial distribution   **(b) $H_{\leq d_1}$ and $H_{\leq d^*}$**

Figure 1: Running example

- – We propose a polynomial algorithm for MKCC checking, in which we reduce MKCC checking to min cut problem in a proposed flow network. **(Section** 6)
- Extensive experiments: we conduct extensive experiments on real datasets to demonstrate the efficiency of the proposed algorithm and the effectiveness of the proposed geo-social group model. **(Section** 7)

2 PROBLEM FORMULATION

We first formulate MKCSSG model and MKCSSG search problem.
Data. We consider an undirected graph data $G = (V, E)$ with network structure, spatial attribute and textual attribute . G has a set of vertices (users) V and a set of edges (friendships) E. For each vertex $v \in V(G)$, v has a piece of location information expressed as latitude and longitude denoted as $(v.x, v.y)$, and has a set of keyword attributes denoted as $v.A$.

We formally define the query for searching MKCSSG.
Query for MKCSSG. The query Q for MKCSSG consists of a social parameter c (an integer), a set of keywords φ, keyword parameters P (a set of integers), a keyword capacity parameter r (an integer), and a location λ (latitude and longitude).
Social constraint. We consider minimum trussness [6] to measure the social cohesiveness of an MKCSSG $S \subseteq G$. Trussness is defined based on the number of triangles each edge is involved in a graph. In general, given a subgraph $S \subseteq G$, we use $\triangle_{uvw}$ to denote a triangle consisting of vertices $u, v, w \in V(S)$.
Support. The support of an edge $e(u, v) \in E(S)$, denoted by $sup(e, S)$, is the number of triangles containing e, i.e., $sup(e, S) = |\{\triangle_{uvw} : w \in N(v, S) \cap N(u, S)\}|$, where $N(v, S)$ and $N(u, S)$ are the neighbours of u, v in S correspondingly.
Minimum subgraph trussness. The trussness for a subgraph S is defined as an integer c that is 2 plus the minimum possible support for edges in $E(S)$. That is, the minimum subgraph trussness defines that for every edge $e \in E(S)$, the number of triangles in which e participates shall be no less than c - 2.

Based on the definition of trussness, we define the c-truss constraint of an MKCSSG S as follows:
Definition 2.1. c-truss constraint. *An MKCSSG S satisfies c-truss constraint if the trussness of S is c, and S is connected.*

Intuitively, if S satisfies c-truss constraint, the vertices of an edge in S have at least c-2 common neighbours in the group S, every vertex in S has no less than c-1 neighbours and at least c-1 edges have to be deleted in order to make S disconnected. The communication cost of S is at most $\lfloor \frac{2|V(S)|-2}{c} \rfloor$. An S with a large value c indicates strong internal social relationships over vertices.
Minimum keyword and capacity constraints. Given a set of query keywords $\varphi = \{k_1, \ldots, k_{|\varphi|}\}$, $P = \{\rho_1, \ldots, \rho_{|\varphi|}\}$, r, and S, MKCC is formally defined below.
Definition 2.2. *Minimum keyword and capacity constraints, MKCC*. *S satisfies MKCC if there is a $v.A' \subseteq \varphi$ for every $v \in V(S)$ such that:*

- *Capacity constraint: $|v.A'| \leq r$,*
- *Minimum keyword constraint (MKC): $\forall\ k_i \in \varphi$, $|V(S_{k_i})| \geq \rho_i$, where $V(S_{k_i})$ is the set of vertices such that for each $v \in V(S_{k_i})$, $v.A'$ contains $k_i \in \varphi$.*

Searching objective. Now, we formalize the spatial closeness for MKCSSG and the research problem.
Spatial closeness. Given a query location λ, we consider a distance function to measure the closeness between λ and an MKCSSG S as:
Definition 2.3. *Spatial closeness*.

$$dist(\lambda, S) = max\{\|\lambda - v\| | v \in V(S)\},$$

where $\|\lambda - v\|$ denotes Euclidean distance between v and λ.
Definition 2.4. (P, c, r, d)**-truss**. *Given a $Q = \{\lambda, P, \varphi, c, r\}$ and a distance threshold d, a subgraph $S \subseteq G$ is a (P, c, r, d)-truss, if it satisfies all the conditions below.*

- *S satisfies MKCC.*
- *S satisfies c-truss constraint.*
- *$dist(\lambda, S) \leq d$.*

Research Problem. MKCSSG search. *Given a query $Q = \{\lambda, P, \varphi, c, r\}$ and G, return a (P, c, r, d)-truss S^* so that there is no (P, c, r, d')-truss S' with $d' \leq d$.*

Example 2.1. An example dataset is shown in Figure 1, where Figure 1 (a) shows locations for vertices of graph data in Figure 1 (b). Let the query be: $Q = \{\lambda, P = \{2, 2, 2\}, \varphi = \{k_1, k_2, k_3\}, c = 4, r = 1\}$. $\{d, e, f, g, h, i\}$ induced subgraph S^* is the optimum result for Q for this dataset. S^* satisfies social constraint, i.e., every edge in $E(S^*)$ involves no less than 2 triangles. S^* satisfies MKCC. That is, it firstly satisfies capacity constraint, i.e., every vertex contributes to at most one keyword in φ, where $d.A' = \{k_1\}$, $e.A' = \{k_2\}$, $f.A' = \{k_3\}$, $g.A^{'} = \{k_1\}$, $h.A' = \{k_2\}$, $i.A' = \{k_3\}$. Then it satisfies MKC, i.e., with the A' for each vertex (those underlined), the keyword vertex frequency for every query keyword is no less than 2. Last but not least, among all groups satisfying the constraints, S^* is the closest one to λ and the most distant vertex (to λ) in S^* is f.

Notice. For the purpose of brevity, in Sections 3, 4 and 5, we first simplify MKCC to MKC and assume every integer in P has an equal value of ρ. In Section 6, we discuss general MKCC checking with r and $P = \{\rho_1, \ldots, \rho_{|\varphi|}\}$.

3 SEARCH FRAMEWORK

Before showing the search framework, we firstly introduce a pre-pruning technique and some definitions.
Maximal (ρ, c)-truss based pruning. A (ρ, c)-truss is a (ρ, c, r, d)-truss, ignoring constraints induced by r and d. A maximal (ρ, c)-truss is a (ρ, c)-truss that cannot be extended by adding either an edge or a vertex.

Algorithm 1: SEARCHMKCSSG(Q,H)

```
Output: $S^*$
1 $d \leftarrow$ initial search distance for $H_{\leq d}$ ;
2 $S^* \leftarrow \emptyset$ ;
3 $S \leftarrow$ ISPCTRUSSIN($H_{\leq d}$) ;
4 while $S$ is $\emptyset$ and $H_{\leq d} \neq H$ do
5     $H_{\leq d'} \leftarrow$ NEWRANGE($d$);
6     $S \leftarrow$ ISPCTRUSSIN($H_{\leq d'}$ );
7     $S^* \leftarrow S, d \leftarrow d', H_{\leq d} \leftarrow H_{\leq d'}$ ;
8 $S^* \leftarrow$ REDCUEPCTRUSS($S^*$);
9 return $S^*$;
```

Given an MKCSSG query containing parameters ρ and c, it is clear that MKCSSG for the query can only reside in a maximal (ρ, c)-truss if it exists. As such, given query, computing maximal (ρ, c)-truss subgraphs contained in G reduces the search space significantly. This can be done by traversing maximal c-truss subgraph with the state of the art truss technique [24].

Definition 3.1. **d radius bounded graph**. *Given a query location λ, a subgraph H and a distance threshold d, d radius bounded graph, denoted as $H_{\leq d}$, is the subgraph of H induced by vertices of H with distance to λ no greater than d.*

We would like to highlight an instance of d radius bounded graph, d^* radius bounded graph, $H_{\leq d^*}$. $H_{\leq d^*}$ has the property below. There is no $H_{\leq d'}$ such that $H_{\leq d'}$ contains MKCSSG and $d' < d^*$.

Optimum search space. We refer $H_{\leq d^*}$ as optimum search space since it is just large enough to contain MKCSSG for the query.

For instance, in Figure 1, $H_{\leq d_1}$ and $H_{\leq d^*}$ are demonstrated. d_1 and d^* identified regions are displayed in Figure 1(a), i.e., cycles centred by λ with radius of d_1 and d^* respectively. The subgraphs are shown in Figure 1(b), i.e., $H_{\leq d_1}$ is the subgraph in grey coloured area and $H_{\leq d^*}$ is the subgraph in doted area. $H_{\leq d^*}$ is the optimum search space containing MKCSSG for the query in Example 2.1.

Next we show the search framework for MKCSSG. It firstly approaches to an $H_{\leq d'}$ just sufficient large to constrain $H_{\leq d^*}$ quickly. Then it reduces $H_{\leq d'}$ to the optimum result.

The framework. As shown in Algorithm 1, MKCSSG search framework consists of two stages: expanding stage (lines 3 to 7) and reducing stage (line 8). During the expanding stage, Algorithm 1 intends to quickly identify $H_{\leq d}$ that is just sufficiently large to contain the optimum search space $H_{\leq d^*}$ by exploring $H_{\leq d}$ that progressively gets larger, in which ISPTTRUSSIN is called to determine the existence of a subgraph satisfying all constraints. For the reducing stage, to get the optimum result, REDUCEPCTRUSS attempts to progressively remove the vertex that is the most distant to λ in S^*. The last survived (ρ, c)-truss during the vertices removing process is the optimum result.

In the following sections, we will discuss details of the two stages. Due to space limitations, proofs for lemmas are omitted while detailed algorithms are in the appendix. We will propose techniques that make the expanding stage have the time complexity of one-time calling of ISPTTRUSSIN. For the reducing stage, we will propose a novel online index and combine the index with our proposed reducing strategy to efficiently check all constraints of MKCSSG. Eventually, our proposed techniques can guarantee that Algorithm 1 has a time complexity of one time truss computation for MKCSSG with MKCC.

4 EXPANDING STAGE

In this stage, we explore a set of d radius subgraphs, starting from a relatively small d radius subgraph and stopping at the first d radius subgraph that is a supergraph of $H_{\leq d^*}$.

Challenges. Since the expanding stage involves expensive constraints checking, our first challenge is how to devise an expanding strategy that can elegantly bound the overall computations tightly. On the other hand, if we can expand to d^* with fewer attempts, the performance will be improved. This can be achieved by starting the search from a d radius graph with d that is close to but no greater than d^*. This arises the second challenge: can we identify such initial search range efficiently? At last, when processing an $H_{\leq d}$ during the expanding stage, if we apply multi-constraint checking just on some restricted subgraphs of $H_{\leq d}$ that potentially contain a (ρ, c)-truss, the search performance can be further boosted. This arises the third challenge on how to quickly identify those potential subgraphs in $H_{\leq d}$?

In the following sub-sections, we will address the three challenges consecutively.

4.1 Expanding Strategy

In this part, we propose an expanding strategy which can bound the total size of subgraphs that will be evaluated.

We first define an expanding invariant as follows.

Definition 4.1. **Δ *size invariant***. *Let $\{d_1, d_2, \ldots, d_i\}$ be the series of radius for defining d radius graphs, for any two consecutive d, d', we define Δ invariant as $\Delta = \frac{|E(H_{\leq d'})|}{|E(H_{\leq d})|}$, in which $\Delta > 1$ must hold.*

The strategy. The strategy applied for the expanding stage is to maintain Δ size invariant over any two consecutively evaluated $H_{\leq d}$, $H_{\leq d'}$. Applying Δ invariant for the expanding stage guarantees two nice properties below.

Property 4.1. ***Nearest first search***. *Vertices accessed by the expanding stage are in non-increasing order according to their distance to λ on a batch basis.*

Property 4.2. ***Power law expansion*** *[3]. The sizes of the set of d radius graphs follow power law expansion, i.e., $\{H_{\leq d_1}, \ldots, H_{\leq d_i}\}$ equals $\{|H_{\leq d_1}|\Delta^0, \ldots, |H_{\leq d_1}|\Delta^{i-1}\}$.*

The two properties help us introduce and prove a lemma as follows.

Lemma 4.1. *Let $H_{\leq d_{i-1}}$, $H_{\leq d_i}$ be the last two d radius subgraphs evaluated by the expanding stage, we have $|E(H_{\leq d_{i-1}})| < |E(H_{\leq d^*})| < |E(H_{\leq d_i})|$.*

The correctness is clear. Firstly, when expanding, Properties 4.1 and 4.2 hold. Secondly, the expanding stage stops when H_{d_i} is the *first* d radius subgraph containing a (ρ, c)-truss.

Next, we establish precise relationship between $|E(H_{\leq d^*})|$ and $|E(H_{\leq d_i})|$ via the lemma below.

Lemma 4.2. *Let $H_{\leq d_i}$ be the last d radius subgraph evaluated by the expanding stage, the inequality $\frac{|E(H_{\leq d_i})|}{|E(H_{\leq d^*})|} < \Delta$ holds.*

Now, let us show the tight bound that is guaranteed by applying the proposed expanding strategy.

Lemma 4.3. *Let $\{H_{\leq d_1}, \ldots, H_{\leq d_i}\}$ be the set of d radius subgraphs evaluated in order by the expanding stage, the inequality $\sum_{j=1}^{i} |E(H_{\leq d_j})| \leq (1 + \frac{1}{\Delta - 1})|E(H_{\leq d_i})|$ must hold.*

Discussion. With Lemma 3, the correctness of the following statement is clear. The running time of lines 3 to 8 in Algorithm 1 is proportional to $(1 + \frac{1}{\Delta^a - 1}) \times \Delta^a \times$ the time complexity of ispcTrussIn($H_{\leq d^*}$), where a is determined by the time complexity of ispcTrussIn($H_{\leq d^*}$) (later on we show a equals 1.5 for MKCSSG with MKC and equals 3 for MKCSSG with MKCC). This provides a tight bound for the expanding stage if we access every $H_{\leq d}$ locally during the loop of lines 3 to 8. As such, we will introduce techniques that ensure local explanation during the expanding stage.

Lemma 4.4. *For any maximal connected (ρ, c)-truss H and fixed query, there is a data structure that takes $O(|E(H)|)$ space, that can be built in $O(|V(H)| \log(|V(H)|))$ time, and that retrieves $E(H_{\leq d})$ in $O(|E(H_{\leq d})|)$ time.*

With the data structure, for consecutive evaluated d and d', we can retrieve $H_{\leq d'}$ based on $H_{\leq d}$ with time linear to $|E(H_{\leq d'}) \setminus E(H_{\leq d})|$.

4.2 Initial Expanding Range

Intuitively, if the initial search range is close to d^*, the total size of subgraphs that has to be evaluated to approaching $H_{\leq d^*}$ is small. This motivates us to find a lower bound of d radius subgraph.

We define a lower bound d radius subgraph, denoted as $H_{\leq \underline{d}}$ as follows.

Definition 4.2. *$H_{\leq \underline{d}}$. A subgraph $H_{\leq \underline{d}}$ of H is a lower bound d radius subgraph of $H_{\leq d^*}$ if it satisfies conditions: 1) $H_{\leq \underline{d}}$ is connected, 2) $H_{\leq \underline{d}}$ satisfies minimum keyword constraint and 3) there is no $H' \subseteq H_{\leq \underline{d}}$ such that H' satisfies the first two constraints and $dist(\lambda, H') < dist(\lambda, H_{\leq \underline{d}})$.*

$H_{\leq \underline{d}}$ relaxes the structure constraint of MKCSSG. As such, it can be computed efficiently, discussed in the lemma below.

Lemma 4.5. *There is an algorithm, Algorithm 2 in the appendix using refined union-find structure, finding $H_{\leq \underline{d}}$ with time complexity of $O(\alpha(|V(H_{\leq d^*})|)|E(H_{\leq d^*})|)$, where $\alpha(|V(H_{\leq d^*})|) \leq 5$.*

4.3 Checking (ρ, c)-truss in d Radius Subgraph

In this section, we show the details of checking (ρ, c)-truss in a d radius subgraph $H_{\leq d}$, the procedure isptTruss in Algorithm 1.

To simplify the discussion, for any two consecutive $H_{\leq d}$ and $H_{\leq d'}$ with $\frac{|H_{\leq d'}|}{|H_{\leq d}|} = \Delta$, let us introduce a new notation $H_{d' \setminus d}$ to denote the subgraph of $H_{\leq d'}$ induced by vertices appearing in edges of $E(H_{\leq d'}) \setminus E(H_{\leq d})$.

We will propose two techniques to speed up (ρ, c)-truss checking. The first increases the performance by restricting the size of subgraphs which we perform truss computation on. The second aims to further restrict the size of subgraphs that performs keyword and connectivity constraints checking.

Lazy (ρ, c)-truss checking strategy. Given $H_{\leq d}$, we only apply (ρ, c)-truss checking on any subgraph potentially containing (ρ, c)-truss, defined as ρ potential subgraph below.

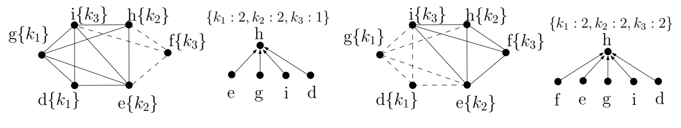


Figure 2: Truss union

ρ potential subgraph $P_{\leq d}$. A subgraph $P_{\leq d} \subseteq H_{\leq d}$ is defined as ρ potential subgraph if it is connected, satisfies minimum keyword constraint and is *maximal* within $H_{\leq d}$.

The strategy. Since a (ρ, c)-truss should reside in $P_{\leq d}$, we propose lazy (ρ, c)-truss checking strategy that applies (ρ, c)-truss constraint checking on every $P_{\leq d}$ in $H_{\leq d}$ only instead of the entire $H_{\leq d}$.

Identifying all $P_{\leq d}$ can be done almost at no cost by using keyword aware union-find structure. That is, when expanding $H_{\leq d}$ to $H_{\leq d'}$, vertices in edges of $H_{\leq d'}$ are progressively added to keyword aware union-find structure. As such, the ρ potential subgraphs in $H_{\leq d'}$ can be retrieved easily since every set in keyword aware union-find structure satisfying minimum keyword constraint identifies a ρ potential subgraph.

Please note that the computation discussed below shall be performed on ρ potential subgraphs only. The size of these subgraphs is vastly restricted compared to the size of $H_{\leq d}$.

Union with existing truss. To avoid graph traversing for checking minimum keyword constraint and connectivity after updating trussness, we propose a solution below. Firstly, we maintain every maximal connected c truss subgraph in every $P_{\leq d}$, each of which is attached with keyword vertex frequency. Secondly, after $P_{\leq d}$ is expanded to $P_{\leq d'}$, we update the maintained c-truss subgraphs if applicable. Although this approach cannot update trussness for existing truss subgraphs precisely, it is sufficient and efficient to check the existence of (ρ, c)-truss in $P_{\leq d'}$. As such, minimum keyword constraint and connectivity checking for truss subgraphs can be performed simultaneously and incrementally. We give formal explanations below and focus on truss unions for expanding a $P_{\leq d}$ to $P_{\leq d'}$. Since all $P_{\leq d}$ in $H_{\leq d}$ are disjoint, the truss union for expanding a $P_{\leq d}$ to $P_{\leq d'}$ can be easily extended to truss unions for expanding $H_{\leq d}$ to $H_{\leq d'}$.

Existing truss $C_{\leq d}$. We maintain connected c-truss subgraphs $C_{\leq d} \subseteq P_{\leq d}$ if they exist. For each $C_{\leq d} \in C_{\leq d}$, its keyword vertex frequencies for every keyword in φ are recorded.

Truss potential subgraph. After expanding $P_{\leq d}$ to $P_{\leq d'}$, we only compute maximal truss subgraphs in *truss potential subgraph* defined below.

Definition 4.3. ***Truss potential subgraph.*** *Given two consecutive $P_{\leq d} \subseteq H_{\leq d}$ and $P_{\leq d'} \subseteq H_{\leq d'}$ with $C_{\leq d} \subseteq P_{\leq d}$, the truss potential subgraph is defined as $TP_{d \setminus d'} = H_{\leq d'}(V')$, where V' is the set of vertices appearing in $E(P_{\leq d'}) \setminus E(C_{\leq d})$.*

The sufficiency of $TP_{d \setminus d'}$ is clear since it contains all triangles in $P_{\leq d'}$ for edges that are not in $C_{\leq d}$ but potentially lead to (ρ, c)-truss.

Truss union. Based on Definition 4.3, for consecutive $P_{\leq d}$ and $P_{\leq d'}$, we compute maximal truss subgraphs in $TP_{d \setminus d'}$ and then add them to $C_{\leq d}$ via union operation, which forms $C_{\leq d'}$.

Example. In Figure 2, we show an example for truss union operation. In Figure 2(a), let $\{g, d, e, i, h\}$ induced subgraph be $H_{\leq d}$

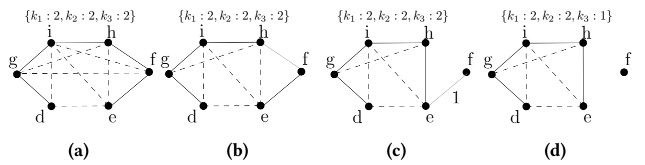

$\{k_1:2, k_2:2, k_3:2\}$
$\{k_1:2, k_2:2, k_3:2\}$
$\{k_1:2, k_2:2, k_3:2\}$
$\{k_1:2, k_2:2, k_3:1\}$
i
h
g
f
d
e
1
(a)
(b)
(c)
(d)

Figure 3: keyword aware spanning forest

and its $P_{\leq d}$ and $C_{\leq d}$ are the same graph. Let the whole graph in Figure 2(a) be $H_{\leq d'}$. Then, $P_{\leq d'}$ is $\{g, d, e, i, h, f\}$ induced subgraph, and $E(P_{d'}) \setminus E(C_{\leq d})$ is $\{ (f, e), (f, h), (f, i) \}$. Then $TP_{d' \setminus d}$ is $\{i, h, f, e\}$ induced subgraph shown in Figure 2(b). Since there is a c-truss in $\{i, h, f, e\}$ induced subgraph, truss-union data structure in Figure 2(a) is updated to the one in Figure 2(b).

Lemma 4.6. *There is an algorithm, Algorithm 3 in the appendix, that can perform (r, c)-truss checking for $H_{\leq d}$ with time complexity of $O(|E(H_{\leq d})|^{1.5})$.*

To conclude the expanding stage, we show lemma below.

Lemma 4.7. *The time complexity of expanding stage is $O\left((1+\Delta^{1.5}+\frac{1}{\Delta^{1.5}-1}) \times |E(H_{\leq d^*})|^{1.5}\right)$.*

The correctness is clear based on the time complexity of Algorithm 3 and Lemma 4.3. When $\Delta = 2$, the time complexity becomes the minimum that is $O(|E(H_{\leq d^*})|^{1.5})$.

5 REDUCING STAGE

For the reducing stage, we focus on searching MKCSSG in the (ρ, c)-trusses found by the expanding stage, denoted as S. We would like to revisit that the size of S is $O(|H_{\leq d^*}|)$.

Intuitively, this stage progressively removes the vertex in S that is most distant to the query location till there is no (ρ, c)-truss in the remaining S. The last survived (ρ, c)-truss is MKCSSG.

Efficiently checking the existence of (ρ, c)-truss after deleting a vertex is challenging. This is because after a vertex deletion, we have to deal with truss computation, verifying minimum keyword constraint and checking connectivity. The obvious time consuming part is truss computation, which can be bounded nicely by taking the advantage of decremental truss computation. The pitfall when analyzing the cost is ignoring the cost of minimum keyword constraint and connectivity checking. We will propose efficient approach for checking multiple constraints together.

5.1 Reducing Strategy

In this part, we show the reducing strategy.

The strategy. We progressively remove the vertex that is most distant to λ (the query location) in S and check the existence of (ρ, c)-trusses in the remaining of S after the deletion. If there exists one, we continue to delete the next most distant vertex in S. Otherwise, We return the last (ρ, c)-truss as MKCSSG.

Clearly, the strategy can find MKCSSG in S correctly since it maintains an invariant that every time deleting the most distant vertex in S, S contains a set of (ρ, c)-trusses. Detailed algorithm is shown in Algorithm 4 in the appendix.

Using the strategy, we can bound the truss constraint checking cost. Next, we propose techniques for efficiently performing keyword-aware connectivity checking.

5.2 Keyword-aware Connectivity Checking

In this section, we show how to efficiently check the existence of a connected subgraph satisfying minimum keyword constraint after an edge is deleted induced by removing the most distant vertex.

High level idea. We will maintain a minimum spanning forest for S (the result of the expanding stage) augmented with aggregated keyword vertex frequency. Notice that initially, every spanning tree in the forest satisfies minimum keyword constraint. After an edge is deleted from S, one of the two cases below may happen.

Case 1: the deleted edge is not in the forest. In this case, the remaining subgraphs are still connected and each connected subgraph still satisfies minimum keyword constraint.

Case 2: the deleted edge is in the forest. In this case, one of the tree in the minimum spanning forest is cut into two trees, which may lead to one of the following subcases.

Subcase 1: cannot link the cut trees. In this subcase, we cannot find a replacement edge from the remaining S to link the two trees, which means the subgraph referred by the two trees becomes two disjoint subgraphs. We update keyword vertex frequency for each of the cut tree. After the update, we safely prune the cut tree from the maintained spanning forest if it does not satisfy minimum keyword constraint since they cannot contribute to MKCSSG.

Subcase 2: can link the cut trees. If we can find a replacement edge, the subgraph referred by two cut trees is still connected. We link the two trees with the replacement edge. Keyword vertex frequency remains the same.

It is clear that the above idea can correctly maintain all connected subgraphs satisfying minimum keyword constraint if they exist after deleting an edge from S. But, it is challenging to preform the maintenance efficiently since checking the existence of a replacement edge could be costly.

To make the maintenance efficient, we borrow the idea from [11]. Given S, every edge in $E(S)$ is associated with a level progressively increased as edges are deleted, which is equivalent to progressively partitioning S hierarchically. Edges with high level refer to a more restricted part of S. In contrast, edges with low level refer to a more general part of S (super graphs of the high level subgraphs). As such when deleting an edge with a certain level, we do not need to consider any edge with lower level as a replacement edge, which elegantly reduces the search space for finding a replacement edge.

We use an example to demonstrate our method.

Example. Suppose we have the input graph as shown in Figure 3(a) with keywords the same as Figure 2(a), and we want to remove f. The minimum spanning forest is shown in Figure 3(a) with edges in solid lines and the edges not in the spanning forest are shown as dashed lines. We do not show the level of an edge if its level is 0. Removing f is equivalent to remove edges incident to f. It is trivial to remove (i, f) and (g, f) since they are not a part of the spanning forest. After that, supposing that we remove (h, f) shown as grey line in Figure 3(b), the spanning tree becomes two trees where the tree with vertices of $\{f, e\}$ is the smaller tree and the level of the edge in the tree is increased by 1. By checking edges incident to f and e, we find a replacement edge (h, e). By connecting the two trees, the spanning tree becomes the one in Figure 3(c). Next, we remove (e, f) shown in Figure 3(c) and the tree with vertex only f is the smaller tree. In this case, we cannot find any edge incident to f,

Table 1: Parameter settings

Parameter	Range	Default value
c	3, 4, 5, 6, 7, 8	6
$\lvert\varphi\rvert$	1, 3, 5, 7, 9	3
$\rho = min\{P\}$	1, 3, 5, 7, 9	3

Table 2: Statistic information in datasets

Dataset	#vertices	#edges	#checkins	c_{max}
Foursquare	4,899,219	28,484,755	1,021,970	16
Weibo	1,019,055	32,981,833	32,981,833	11
Yelp	257,532	957,711	431,563	21

leading to Figure 3(d). We know that the graph becomes separated and we also know that there is a connected component in the remaining graph with keyword frequencies of $\{k_1{:}2,\ k_2{:}2,\ k_3{:}1\}$. Without using the proposed method, we cannot simultaneously know the keyword vertex frequency and the connectivity of the subgraph after deleting f.

The detailed algorithm for keyword-aware connectivity checking is in the appendix. The time complexity of keyword-aware connectivity checking for deleting $|E(H_{\leq d^*})|$ number of edges is $O(|E(H_{\leq d^*})|(log_2|V(H_{\leq d^*})|)^2)$. The proof idea is similar to [11].

6 MKCC CHECKING

In this section, we propose a polynomial algorithm for minimum keyword and capacity constraints (MKCC) checking.

We will show that MKCC checking for a set of vertices S, query keywords $\varphi = \{k_1, \ldots, k_{|\varphi|}\}$, $P = \{\rho_1, \ldots, \rho_{|\varphi|}\}$ and r can be reduced to an instance of min cut problem, where a min cut problem is: given a flow network N, partition its nodes into S and T parts so that the sum of the capacities across S and T is minimized. More preliminaries for min cut problems are omitted due to the limited space. Please be noticed, we use node to denote vertex in N to differentiate the vertex in the input graph.

The instance of maximum flow problem. We construct the flow network N based on φ, P, r and S. N consists of four types of nodes below. For each keyword in φ, we create a keyword node. For each vertex in S, we create a vertex node. Additionally, we create a source node s and a sink node t. The edges and capacities for N are as follows. For each vertex node n, we create an edge from s to n with capacity of r. For each keyword node n' representing k_i, we create an edge from n' to t with capacity of ρ_i. In addition, there is an edge from a vertex node n to a keyword node n' if the keyword attributes of n representing vertex contain the query keyword represented by n', and the capacity between n and n' is set to ∞.

Lemma 6.1. *S satisfies MKCC if there exists a min cut for N whose T part contains node t only.*

We adopt preflow–push (push-relabel) algorithm to solve the min cut problem. As such, the time complexity of MKCC checking for MKCSSG search is shown below.

MKCC checking complexity for the expanding stage. This part can be bounded by $O\left((1+\Delta^3+\frac{1}{\Delta^3-1})\times|V(H_{\leq d^*})|^3\right)$, assuming $|\varphi| \ll |V(H_{\leq d^*})|$. As discussed previously, by letting $\Delta = 2$, the time complexity becomes minimum, $O\left(|V(H_{\leq d^*})|^3\right)$.

MKCC checking complexity for the reducing stage. This part can be bounded by $O\left(|V(H_{\leq d^*})|^3\right)$ as well, by taking advantage of the preflow-push algorithm. For the reducing stage, when deleting a vertex, we do not reconstruct the flow network and compute the min cut for the new constructed flow network from scratch. Instead, we set the capacity of the edge (from s to the deleted vertex) to 0. As such, in the capacity-refined flow network, the flow becomes a preflow again, while the labelling information in the previous computation can be reused. Taking advantage of such nice property, solving min-cut problem for such successive flow networks can be done with time complexity that is equivalent to solving one of the min-cut problem [9]. This technique is known as parametric flow network. The reduction details, i.e., from finding a min cut for our progressively refined N to the parametric flow network problem, are omitted due to space limitations.

7 EXPERIMENTAL STUDY

In this section, we conduct experimental studies on real datasets to evaluate the proposed model and algorithms.

Datasets. For effectiveness evaluation, we conduct the experiments on an open dataset Yelp with real textual, social and spatial information. Textual information on Yelp are pre-processed using Porter stemming algorithm, to increase the keyword matching rate. The location of each user is derived from their latest check-in store. For efficiency evaluation, we conduct the experiments over two large open datasets including Foursquare and Weibo. For these two datasets, each social user contains some check-in locations. Since we only need one check-in for each user, we select the latest check-in as the spatial coordinate. The keyword attribute of each user is randomly assigned for the first two datasets in Table 2 using the method similar to [12]. That is, keyword pool is generated firstly. After that, we assign each vertex with 0-7 of the keywords randomly. Table 2 presents the statistics for all datasets.

7.1 Effectiveness Evaluation

To demonstrate the effectiveness of MKCSSG search with MKC and MKCC, we analyze 2 representative queries on Yelp dataset.

Queries. Q_1. The first query we evaluated has parameters as follows: $\lambda = (36.11, -115.13)$ $\varphi = \{salad, chicken, beef, BBQ\}$, $P = \{10, 10, 10, 10\}$, $r = 4$, $c = 5$. This query can be used to find participants around Las Vegas to attend a BBQ party providing foods such as beef and chicken, assuming textual information on each user reflects the user's diet favourites.

Q_2. The second query evaluated has parameters: $\lambda = (36.11, -115.13)$ $\varphi = \{guitar, piano, violin\}$, $P = \{2, 1, 2\}$, $r = 1$, $c = 4$. This query intends to form a band to perform around Las Vegas, assuming textual information on each user reflects the user's skills.

Compared models. Except for our models, the representative geo-social group model proposed in [25] and its adaption are tested.

Minimum acquaintance geo-social group. We use the instance of the model proposed in [25] that has no size constraint as baseline. This model adopts minimum degree as social constraint and does not consider keyword cohesiveness, denoted as MASG.

Minimum acquaintance, collective keyword coverage geo-social group. We adapt MASG to make it sensitive to query keyword by adding a keyword constraint that keyword attributes of the members in the found geo-social group shall cover all query keywords collectively. This model is denoted as MACSG.

Evaluation metrics. Let S denote the found result by one of the compared models. We report the results below.

Table 3: Effectiveness evaluation

Query	Model	ED	MC	KVP	MKVP	Dist
Q_1	MASG	0.34	0.77	0	0	*0.1*
	MACSG	0.38	0.71	0.12	0.03	0.13
	MKCSSG with MKC	0.51	0.45	0.67	0.44	0.14
	MKCSSG with MKCC	**0.61**	**0.41**	**0.81**	**0.51**	0.18
Q_2	MASG	0.27	0.67	0	0	*0.12*
	MACSG	0.38	0.56	0.11	0.02	0.56
	MKCSSG with MKC	0.61	0.32	0.45	0.33	0.59
	MKCSSG with MKCC	**0.68**	**0.28**	**0.62**	**0.42**	0.61

Social cohesiveness. We report edge density (ED), i.e., $\frac{|E(S)|}{|V(S)|}$ and the maximum communication cost [19] (MC), i.e., the length of the longest shortest path between any two vertices in S, to show the social cohesiveness of the found results.
Relevance to query keywords. We report keyword vertex percentage (KVP) and the keyword vertex percentage for the keyword appears least (MKVP) to show the keyword relevance.
Spatial closeness. We report the spatial distance (Dist) of S to the query location λ, defined in Definition 2.3.

All the above results are normalised to a value between 0 to 1 and are reported in Table 3. Except for spatial closeness and maximum communication cost, results with higher score are superior. Overall speaking, for both Q_1 and Q_2, MKCSSG with MKCC outperforms other models substantially and MKCSSG with MKC is the runner-up. From keyword cohesiveness perspective, the results justify that the proposed keyword constraints can find geo-social group that are related to query keywords. From social cohesiveness perspective, using c-truss constraint would find geo-social groups with much less communication cost compared to models using minimum degree constraint. Compared to other models, MKCSSG with MKC or MKCC can find much more effective group from social and keyword cohesiveness perspectives while just compromising spatial closeness marginally.

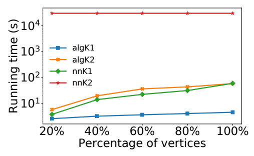
Due to the limited space, a case study is shown in the appendix.

7.2 Efficiency Evaluation

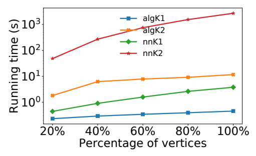
Evaluated algorithms. In the experiments below, we denote Algorithm 1 applying MKC and MKCC as algK1 and algK2. Besides, we also adapt nearest neighbour based search algorithm proposed in [25] with IR-tree index, denoted as nnK1 and nnK2, to solve our problem with MKC and MKCC.

Parameter settings. The experiments are evaluated using different settings of query parameters: c (the minimum truss number), reasonable sets of keywords φ as well as the minimum keyword constraint parameter ρ. The query locations are generated randomly. The ranges of the parameters and their default values are shown in Table 1, in which we select reasonable c based on datasets. Furthermore, when we vary the value of a parameter for evaluation, all the other parameters are set as their default values.

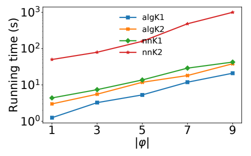
Scalability. To verify the scalability of our algorithms, we choose different sizes of sub-datasets by selecting different percentages of vertices in each dataset. The results are displayed in Figures 4(a) and (b). Overall speaking, algorithms using our proposed search framework (algK1,algK2) are more scalable than nnK1 and nnK2. This is because the proposed search framework has the nice property that can limit the search region while preserving the optimum result. algK1 is most scalable one since it incorporates the proposed techniques and MKC checking is cheap. On the other hand, nnK2 is



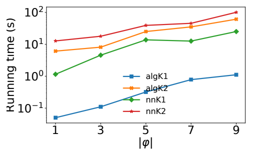
(a) Foursquare



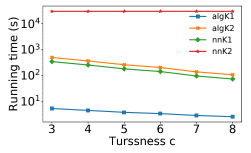
(b) Weibo



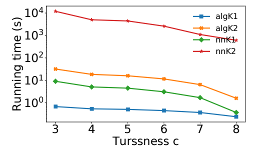
(c) Foursquare



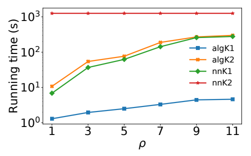
(d) Weibo



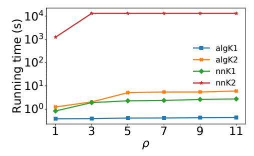
(e) Foursquare



(f) Weibo



(g) Foursquare



(h) Weibo

Figure 4: Efficiency evaluation

the least scalable due to its high time complexity. In large dataset Foursquare, it cannot finish within 4 hours.

Varying $|\varphi|$. Figures 4(c) to (d) demonstrate the running time as $|\varphi|$ varies for different datasets. As the number of query keywords increases, the running time for all the algorithms rise. This is because having more keywords indicates more data need to be explored by those algorithms since they explore vertices from the region near the query location to the region containing the optimum result. For the same keyword constraint schema, our proposed algorithms outperform the NN based algorithms substantially. algK2 can find the optimum result within several seconds when $|\varphi|$ is small while can still answer a query for large $|\varphi|$ within half minute.

Varying c. We evaluate the performance for all the algorithms when varying the trussness c in Figures 4(e) and (f). In general, as c increases, the running time for all the algorithms reduces. The reason is that the size of subgraph with high c tends to be small, which makes search space decrease as c rises. Noticeably, algK1 outperforms the other algorithms consistently. Again, this experiment justifies the superiority of the proposed search framework. algK1 is faster than nnK1and algK2 is much faster than nnK2. nnK2 is the slowest and runs over 4 hours on Foursquare.

Varying ρ. In Figures 4(g) to (h), the running time for the algorithms when we change ρ are shown. For both datasets, the running time of all algorithms increase as ρ increases. For Weibo, algK1 is not very sensitive to the change of ρ. This is because for this dataset the proposed initial search bound can approach the optimum result effectively, and the dominating computation is just trussness verification. Compared to algK2, nnK2 is much slower. This shows the advantage of our proposed search framework and parametric flow network technique.

Table 4: Pruning evaluations

Dataset	$\frac{\lvert H\rvert}{\lvert G\rvert}$	$\frac{\lvert H_{\leq d^*}\rvert}{\lvert H\rvert}$	$\frac{\lvert P_{\leq d^*}\rvert}{\lvert H_{\leq d^*}\rvert}$	$\frac{\lvert C_{\leq d^*}\rvert}{\lvert P_{\leq d^*}\rvert}$
Weibo	58.4%	33.5%	12.4%	5.2%
Foursquare	39.5%	32.2%	10.3%	3.7%

Pruning effectiveness evaluation. We show pruning effectiveness in Table 4 in term of size ratio for corresponding subgraphs evaluated by MKCSSG. The result is the average of 200 randomly generated queries with default settings but different query keywords for each dataset. As we can see, the maximal (ρ, c)-truss based pruning can filter out 40% to 60% of vertices from the original graph. Using the power law expanding, our algorithm only evaluates 20% to 35% of maximal (ρ, c)-trusses for corresponding datasets. It is very noticeable that, our proposed ρ potential and truss potential subgraphs for different datasets are extremely small.

8 RELATED WORKS

Geo-social group discovery. Doytsher et al. [7] combined spatial and social networks and proposed graph-based query processing techniques. Yang et al. [22] considered a socio-spatial group query with the requirement of minimizing the total spatial distance. Armenatzoglou et al. [2] proposed a general framework for geo-social query processing. All these works considered loose social constraints in the query and did not consider diverse demands.

Team formulation. Studies on the formation of teams of socially close experts from a social network have drawn additional research interests recently [13, 14, 19]. However, these studies have mostly focused on minimizing some social metrics in a team without considering the spatial factor.

Spatial-aware community search. In [23], they found (k, r)-core community such that socially the vertices in (k, r)-core is a k-core and from similarity perspective pairwise vertex similarity is more than a threshold r. Recently, three kinds of CS queries have been studied on geo-social networks, namely spatial-aware community search [5, 8] and geo-social group queries with minimum acquaintance constraint [18, 25]. They all required that the communities are structurally and spatially cohesive. But, they did not consider textual cohesiveness w.r.t. a set of query keywords as we did.

Cohesive subgraph search in attributed graph. Recently, attributed community search has drawn great attention such as [4, 12, 16]. They focused on exploring textual and social information while did not consider spatial closeness.

9 CONCLUSION

In this paper, we study geo-social group search with minimum keyword and capacity constraints. We propose a novel search framework making the search towards the optimum result fast. In addition, we propose online data structures, keyword aware union-find structure and keyword-aware forest, and use parametric flow network, which substantially boost the search speed. We also propose heuristics, and truss union operation to further speed up the proposed search algorithm. Extensive experiments are conducted on both semi-synthetic and real datasets, from which the efficiency and the effectiveness are evaluated and justified.

ACKNOWLEDGMENTS

This work is jointly supported by the ARC Discovery Projects under Grant No. DP170104747, DP180100212 and DP200103700, the ARC Linkage Project under Grant No. LP180100750, the Research Grants Council of Hong Kong, China under Grant No. 14203618 and 14202919, and the National Natural Science Foundation of China under Grant No. 61872258.

REFERENCES

[1] Ritesh Ahuja, Nikos Armenatzoglou, Dimitris Papadias, and George J Fakas. 2015. Geo-social keyword search. In *SSTD*. Springer, 431–450.

[2] Nikos Armenatzoglou, Stavros Papadopoulos, and Dimitris Papadias. 2013. A general framework for geo-social query processing. *PVLDB* 6, 10 (2013), 913–924.

[3] Fei Bi, Lijun Chang, Xuemin Lin, and Wenjie Zhang. 2018. An optimal and progressive approach to online search of top-k influential communities. *PVLDB* 11, 9 (2018), 1056–1068.

[4] Lu Chen, Chengfei Liu, Kewen Liao, Jianxin Li, and Rui Zhou. 2019. Contextual community search over large social networks. In *ICDE*. IEEE, 88–99.

[5] Lu Chen, Chengfei Liu, Rui Zhou, Jianxin Li, Xiaochun Yang, and Bin Wang. 2018. Maximum co-located community search in large scale social networks. *PVLDB* 11, 10 (2018), 1233–1246.

[6] Jonathan Cohen. 2008. Trusses: Cohesive subgraphs for social network analysis. *National Security Agency Technical Report* 16 (2008).

[7] Yerach Doytsher, Ben Galon, and Yaron Kanza. 2010. Querying geo-social data by bridging spatial networks and social networks. In *SIGSPATIAL International Workshop on Location Based Social Networks*. ACM, 39–46.

[8] Yixiang Fang, Reynold Cheng, Xiaodong Li, Siqiang Luo, and Jiafeng Hu. 2017. Effective Community Search over Large Spatial Graphs. *PVLDB* 10, 6 (2017), 709–720.

[9] Giorgio Gallo, Michael D. Grigoriadis, and Robert E. Tarjan. 1989. A Fast Parametric Maximum Flow Algorithm and Applications. *SIAM J. Comput.* 18, 1 (Feb. 1989), 30–55.

[10] Bishwamittra Ghosh, Mohammed Eunus Ali, Farhana M. Choudhury, Sajid Hasan Apon, Timos Sellis, and Jianxin Li. 2018. The Flexible Socio Spatial Group Queries. *PVLDB* 12, 2 (2018), 99–111.

[11] Jacob Holm, Kristian De Lichtenberg, Mikkel Thorup, and Mikkel Thorup. 2001. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM* 48, 4 (2001), 723–760.

[12] Xin Huang and Laks V. S. Lakshmanan. 2017. Attribute-driven Community Search. *PVLDB* 10, 9 (2017), 949–960.

[13] Mehdi Kargar, Morteza Zihayat, and Aijun An. 2013. Finding affordable and collaborative teams from a network of experts. In *SDM*. SIAM, 587–595.

[14] Theodoros Lappas, Kun Liu, and Evimaria Terzi. 2009. Finding a team of experts in social networks. In *SIGKDD*. ACM, 467–476.

[15] Yafei Li, Dingming Wu, Jianliang Xu, Byron Choi, and Weifeng Su. 2014. Spatial-aware interest group queries in location-based social networks. *Data & Knowledge Engineering* 92 (2014), 20–38.

[16] Qing Liu, Yifan Zhu, Minjun Zhao, Xin Huang, Jianliang Xu, and Yunjun Gao. 2020. VAC: Vertex-Centric Attributed Community Search. In *ICDE*. IEEE, 937–948.

[17] Weimo Liu, Weiwei Sun, Chunan Chen, Yan Huang, Yinan Jing, and Kunjie Chen. 2012. Circle of friend query in geo-social networks. In *DASFAA*. Springer, 126–137.

[18] C. Shen, D. Yang, L. Huang, W. Lee, and M. Chen. 2016. Socio-Spatial Group Queries for Impromptu Activity Planning. *TKDE* 28, 1 (Jan 2016), 196–210.

[19] Chih-Ya Shen, De-Nian Yang, Wang-Chien Lee, and Ming-Syan Chen. 2016. Spatial-Proximity Optimization for Rapid Task Group Deployment. *TKDD* 10, 4 (2016), 47.

[20] Robert Endre Tarjan. 1975. Efficiency of a Good But Not Linear Set Union Algorithm. *J. ACM* 22, 2 (April 1975), 215–225.

[21] Dingming Wu, Yafei Li, Byron Choi, and Jianliang Xu. 2014. Social-aware top-k spatial keyword search. In *MDM*, Vol. 1. IEEE, 235–244.

[22] De-Nian Yang, Chih-Ya Shen, Wang-Chien Lee, and Ming-Syan Chen. 2012. On socio-spatial group query for location-based social networks. In *SIGKDD*. ACM, 949–957.

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

[24] Yikai Zhang and Jeffrey Xu Yu. 2019. Unboundedness and Efficiency of Truss Maintenance in Evolving Graphs. In *SIGMOD*. 1024–1041.

[25] Qijun Zhu, Haibo Hu, Cheng Xu, Jianliang Xu, and Wang-Chien Lee. 2017. Geo-social group queries with minimum acquaintance constraints. *The VLDB Journal* 26, 5 (2017), 709–727.

A ALGORITHMS

Finding lower bound d radius subgraph. Algorithm 2 demonstrates the major steps for finding $H_{\leq \underline{d}}$. It is a refined union-find process [20]. We augment the union-find data structure with keyword vertex frequency. Algorithm 2 progressively performs union operations on edges in non-increasing order of their distances to λ. By union operations, vertices that are connected are added into the same set. Each set is attached with keyword vertex frequency for each keyword. When an edge (u, v) is being evaluated, Algorithm 2 first finds if u and v are contained in the same set in existing union-find structure (lines 3 to 5). If not, the two sets containing u and v shall be connected via standard union operation and keyword vertex frequency of the two sets shall be aggregated (lines 8 to 9). The discussion for union-find operations is omitted. After a union operation, if there is a set satisfying minimum keyword constraint, we find $H_{\leq \underline{d}}$. Otherwise, Algorithm 2 continues.

The (ρ,c)-truss checking algorithm. The principal steps of (ρ,c)-truss checking are shown in Algorithm 3.

Data structure. Since Algorithm 3 is called iteratively, it works on progressively refined data structures including adjacency list of $H_{\leq d}$, the keyword aware union-find denoted as $UF_{\leq d}$ storing every ρ potential subgraph, the keyword-aware truss union-find structure denoted $TUF_{\leq d}$ storing maximal connected k-truss with aggregated keyword frequency. All those data structures are empty sets before the first time when Algorithm 3 is called.

Principal steps. Algorithm 3 adds each edge in $H_{d' \backslash d}$ to $H_{\leq d'}$ and performs union operation on each edge to $UF_{\leq d'}$, where the edges of $H_{d' \backslash d}$ can be retrieved easily with the sorted array proposed in Lemma 4.4. After that, Algorithm 3 computes maximal c-truss subgraphs in the truss potential subgraph defined in Definition 4.3 (line 5). More precisely, with $UF_{\leq d'}$ and $TUF_{\leq d'}$, $TP_{d' \backslash d}$ is $H_{\leq d'}(V')$, where V' are the vertices appearing in $E(\cup_{P_{\leq d'} \in UF_{\leq d'}} P_{\leq d'}) \setminus E(\cup_{C_{\leq d} \in TUF_{\leq d'}} C_{\leq d})$. Next, Algorithm 3 performs truss union operations for the computed maximal c truss subgraphs. After the truss union, if there is a set in $TUF_{\leq d'}$ that satisfies minimum keyword constraint, then there is a (ρ, c)-truss and Algorithm 3 returns the (ρ, c)-truss S (line 9). Otherwise, Algorithm 3 returns $\emptyset$.

Reducing strategy algorithm. Algorithm 4 shows the major steps of the strategy for the reducing stage. It progressively removes the vertex that is most distant to λ (the query location) in S and checks the existence of (ρ, c)-trusses in the remaining of S after the deletion. If there exists one, Algorithm 4 continues to delete the next most distant vertex in S. Otherwise, it returns the last (ρ, c)-truss as MKCSSG.

Algorithm 2: Finding lower bound search range

```
Input: $H$
Output: $H_{\leq \underline{d}}$
/* W.o.l.g, $DIST(\lambda, u) \leq DIST(\lambda, v)$ */
1  foreach $(u, v) \in$ sorted edge list of $H$ do
2      maintain adjacency list;
3      initialise set rooted as $u$ and $v$ if necessary;
4      $ru \leftarrow$ find(u), $rv \leftarrow$ find(v);
       /* W.o.l.g, $ru.rank \leq rv.rank$ */
5      if $ru \neq rv$ then
6          standard union opertion;
7          $flag \leftarrow true$;
8          foreach $k \in \varphi$ do
9              $ru.k \leftarrow ru.k + rv.k$;
10             if $ru.k < \rho$ then
11                 $flag \leftarrow false$;
12         if flag then
13             $H_{\leq \underline{d}} \leftarrow$ maintained adjacency list;
14             return $H_{\leq \underline{d}}$ ;
```

Algorithm 3: INCISPCTRUSSIN($H_{\leq d}$, d')

```
Input: $H_{\leq d}$, $d'$
Output: $S$
/* $UF_{\leq d}$: keyword aware union find structure storing
   all $\rho$ potential graphs in $H_{\leq d}$ */
/* $UF_{\leq d}$: keyword aware truss union find structure
   storing all truss subgraphs in $H_{\leq d}$ */
1  $H_{\leq d'} \leftarrow H_{\leq d}$, $UF_{\leq d'} \leftarrow UF_{\leq d}$, $TUF_{\leq d'} \leftarrow TUF_{\leq d}$ ;
2  foreach $(u, v) \in H_{d' \backslash d}$ do
3      $H_{\leq d'} \leftarrow H_{\leq d'} \cup \{\{u, v\}\}$;
4      $UF_{\leq d'} \leftarrow UF_{\leq d'} \cup \{(u, v)\}$; // union
   /* $P_{d' \backslash d}$ has been generated during updating $UF_{\leq d'}$ */
5  $H' \leftarrow$ compute $c$-truss in $TP_{d' \backslash d}$;
6  foreach $(u, v) \in H'$ do
7      $TUF_{\leq d'} \leftarrow TUF_{\leq d'} \cup \{(u, v)\}$; // truss union
8  if $TUF_{\leq d'}$ contains a set satisfies keyword constraint then
9      return the set as $S$ ;
10 else
11     return $\emptyset$;
```

Algorithm 4: REDUCEPCTRUSS(Q,S)

```
Input: $S$: $(\rho, t) - truss$
Output: $S^*$
1  sort vertices in $S$ according to their distance to $\lambda$ in none
     decreasing order;
2  foreach $u \in V(S)$ do
3      $S' \leftarrow$ PCTRUSSCECKING($S$, $u$);
4      if $S' \neq \emptyset$ then
5          $S \leftarrow S'$; // order preserved
6      else
7          return $S$ as $S^*$;
8  Procedure pcTrussChecking $(S, u)$
9      $Q \leftarrow \emptyset$;
10     foreach $v \in N(u, S)$ do
11         $Q \leftarrow Q \cup \{(u, v)\}$;
12     while $Q \neq \emptyset$ do
13         $(u, v) \leftarrow Q.pop()$;
14         foreach $w \in N(u, S) \cap N(v, S)$ do
15             update triangle numbers for $(w, u)$, $(w, v)$ ;
16             put $(w, u)$, $(w, v)$ into $Q$ if they cannot be part of
                 $c$-truss;
17             remove $(u, v)$ from $S$ ;
18             if CKCHECKING($(u, v)$) then
19                 return $\emptyset$;
20         return the remaining $S$;
```

Algorithm 5: ckChekcing((u, v))

```
Input: $(u, v)$, $F$
Output: True or False
1  $l = (u, v).level$;
2  if $(u, v) \notin F_{\geq 0}$ then
3      delete $(u, v)$ directly;
4      return True;
5  $T_u, T_v \leftarrow$ delete $(u, v)$ from $F_{\geq l}$;
6  assume $|V(T_u)| \leq |V(T_u)|$;
7  foreach $e \in E(T_u)$ do
8      $e.l \leftarrow e.l + 1$;
9      calculate aggregated keyword vertex frequency in $T_u$;
10 for $i \leftarrow l + 1$ to 0 do
11     cut $F_{\geq i}$ ;
12 $altE \leftarrow \emptyset$;
13 for $i \leftarrow l$ to 0 do
14     for $v \in T_u$ do
           /* $v.adjG_i$ is level-aware adjacency list */
15         for $w \in v.adjG_i$ do
16             if $w \in V(T_u)$ then
17                 increase level of $(v, w)$ by 1;
18             else
19                 $altE \leftarrow (v, w)$, $altE.l \leftarrow i$;
20                 Break ;
21 if $altE \neq \emptyset$ then
22     for $i \leftarrow altE.l$ to 0 do
23         link corresponding two subtrees in $F_i$ via $altE$;
24 else
25     update aggregated keyword frequencies of $T_v$;
26     prune trees in $S$ do not satisfy minimum keyword constraint;
27 return True If at least one of $T_v$ in $F_0$ and $T_u$ satisfies keyword
     constraint else return False;
```

Keyword aware connectivity checking algorithm. Algorithm 5 guarantees that after an edge deletion, every remaining minimum spanning tree in the keyword aware spanning forest satisfies MKC. It returns true if the keyword aware spanning forest is not $\emptyset$. Otherwise it return $\emptyset$. Detailed steps are given below.

Given that (u, v) with level l is to be deleted, Algorithm 5 firstly checks whether it is in the current forest or not.

Case 1: . If (u, v) is not in $F_{\geq 0}$, (u, v) is deleted (line 3), the algorithm returns true .

Case 2: . If (u, v) is in $F_{\geq 0}$, Algorithm 5 deletes it from the tree containing (u, v) from level l which is the highest forest it is in.

Performing tree cut (lines 10 to 11). The tree is cut into two subtrees T_u and T_v, and levels of edges in the smaller tree in terms of number of vertices are increased by 1. Next, Algorithm 5 propagates the deletion from $F_{\geq l+1}$ to $F_{\geq 0}$ so that from the view at all the levels, the tree is split.

Searching a replacement edge (line 13 to 20). After performing tree cut, Algorithm 5 starts to search for a replacement edge of (u, v) that may connect T_u to T_v. This is achieved by checking all the edges incident to the vertices appearing in T_u. To maintain the minimum spanning forest property, Algorithm 5 searches for an alternative edge from level l. If an edge (v, w) is incident to T_u but v and w are in T_u then its level is increased by 1.

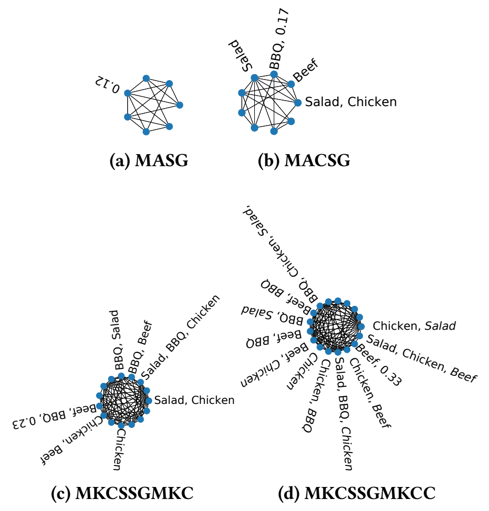


Figure 5: Case study

Subcase 1: cannot link the cut trees (lines 24 to 26). If no replacement edge is found, T_u induced subgraph is isolated. Aggregated keyword frequencies are adjusted. If there is a tree violating the minimum keyword constraint after the adjustment, it is pruned.

Subcase 2: can link the cut tress (lines 21 to 23). If a replacement edge is found, the incident edge (v, w) shall link T_u to T_v and this edge is inserted from $F_{\geq l}$ to $F_{\geq 0}$ so that from the view of all the levels, the tree is linked.

B EXPERIMENTS

Measures. We measure the running time of the algorithms. The reported running time is the total CPU time, excluding the I/O time of loading graph data and indices from disk to main memory, and a timeout of 4 hours is set. All algorithms are implemented in C++. All the experiments are conducted on a PC with CPU of AMD 3900x (12 cores, 24 threads), memory of 128GB DDR4 3600HZ, and Windows 10 (build 1803). All the experiments are conducted no less than 100 times and the average results are demonstrated.

Case study. We demonstrate the results of different models in Figure 5 using a query Q = { (36.11,-115.13), {3,3,3}, {beef, chicken, BBQ, salad}, 4, 1 } on Yelp dataset. It is clear to see that our proposed model can find more cohesive groups (Figures 5(c) and (d)) with members more related to query keywords compared to groups (Figures 5(a) and (b)) using baseline models. From the spatial closeness perspective, our model is not bad, i.e., it sacrifices reasonable spatial closeness to get more rewards on social cohesiveness and keyword relevance w.r.t the query keywords. It also demonstrates the capability of our model for handling demands with diverse specificities. When the capacity of contribution is high ($r \geq 4$), we can find a group of members whose keyword attribute can contribute all the query keywords (Figure 5(c)). When the capacity of contribution is low ($r = 1$), we can find a group with every member just focusing on one query keyword (Figure 5(d) italic keywords are the focused ones) while overall still closely relevant to the query keywords. Figure 5(c) and Figure 5(d) demonstrate that our proposed MKCC is able to find groups according to the demands of real activities.