World Wide Web (2020) 23:2153–2184
https://doi.org/10.1007/s11280-020-00788-w

Personalized top-*n* influential community search over large social networks

Jian Xu[1] · **Xiaoyi Fu**[2] · **Yiming Wu**[1] · **Ming Luo**[1] · **Ming Xu**[1] · **Ning Zheng**[1]

Received: 16 February 2019 / Revised: 24 December 2019 / Accepted: 15 January 2020 /
Published online: 31 March 2020

Abstract
User-centered analysis is one of the aims of online community search. In this paper, we study personalized top-*n* influential community search that has a practical application. Given an evolving social network, where every edge has a propagation probability, we propose a maximal *pk*-Clique community model, that uses a new cohesive criterion. The criterion requires that the propagation probability of each edge or each maximal influence path between two vertices that is considered as an edge, is greater than *p*. The maximal clique problem is an NP-hard problem, and the introduction of this cohesive criterion makes things worse, as it mights add new edges to existing networks. To conduct personalized top-*n* influential community search efficiently in such networks, we first introduce a pruning based method. We then present search space refinement and heuristic based search approaches. To diversify the search result in one pass, we also propose a diversify algorithm which is based on a novel tree-like index. The proposed algorithms achieve more than double the efficiency of the the search performance for basic solutions. The effectiveness and efficiency of our algorithms have been demonstrated using four real datasets.

Keywords Community search · Online · Pruning · Diversified search

1 Introduction

Communities are basic structures for understanding the organization of many real-world networks, such as social, biological, and collaboration networks [17]. Recently,community search over large and evolving graphs has attracted significantly increasing attention. Different from community detection which finds all communities in an entire network, online community search aims to find communities that have certain cohesive relationships w.r.t.

This article belongs to the Topical Collection: *Special Issue on Web Information Management and Applications*
Guest Editors: Yi Cai and Jianliang Xu

✉ Xiaoyi Fu
xiaoyifuu@gmail.com; xiaoyifu@hkbu.edu.hk

Extended author information available on the last page of the article.

the query nodes in an online manner. As the communities for different vertices residing in a network may have very different characteristics, compared with general community detection, the ability for personalized community query, which online community search provides, is more meaningful.

In this paper, we study modeling and querying of the top-n influential communities to a specific query node, termed as personalized top-n influential community search. As the influential communities around a user represent the social contexts for that user, top-n influential community search provides a useful tool for other analytical tasks, such as influential social community discovery and accurate community influence modeling. The following are two examples.

Example 1 (User-centered influential community recommendation) Due to the lack of prior information from users, recommendation for new users is a key challenge and the well-known cold-start problem [29]. As we know, it is a group of users who share similar tastes to be a community. Suppose that Mr. Spike is a Twitter user and would like to join in a music fan community. Sometimes he has been surrounded by many such communities. While there are pop music, classic music and rock'n'roll music communities around him, it is difficult to tell which one influence Spike more than others. Though he does not belong to any music community yet, but one of his friends might have been in a music community. Hence, to learn the influential music communities for Spike, we conduct a top-n influential community search. In this situation, the established communities with high influence to Spike are ideal for recommendation.

Example 2 (Evaluation of community influence) Evaluating community outer influence is typically measured as the number of users influenced by the community or just to reveal the communities with the highest outer influences [22]. In counting users influenced by the community, the criterion is whether the influential probability exceeds a specified value. However, with a top-n influential community search, we have an alternative measurement, which measures the number of users involved in their top-3 influential communities. For example, a music distribution company wants to learn the influence of a music fan community among all the Twitter users. Then the company can estimate how many external users are really influenced by the influential communities to predict the trend of music. The obtained information will help the company to adjust their marketing strategies and enable more scientific decision-making.

The previous examples demonstrate the benefit of the top-n influential community search, there are challenges in this area. The first challenge is how to identify all the communities around a user in the evolving and dynamic social networks. The second is how to calculate the influence of all these communities and finds top-n efficiently. Since there are many overlapping communities, how to diversify the returned results is also challenging.

The foundation of the personalized top-n influential community search is the capability to properly measure community influence to a user. Typically, we model a social network as a graph G with vertices representing individuals and edges representing connections or relationships between any two individuals. According to a stochastic cascade model, such as Independent Cascade (IC) model [14], each edge (u, v) in the network is associated with a propagation probability $w(u, v)$. The influence is evaluated as the probability that the community influences the external user in social networks. In addition to influence measurement, we propose the concept of a maximal pk-Clique community inspired from cliques [2]. A k-clique is a complete subgraph that includes at least k vertices, and is not contained in any other complete subgraph. A maximal pk-Clique community is a community in a social network that has edges between nodes u and v in the graph that can either be original edges in the social network or a path from u to v, and its corresponding maximum influence probability is greater than p. We argue that, with propagation probability of every edge greater than a specific value p, users are well-connected with each other in such a community. A maximal pk-Clique ensures that a discovered community is connected and cohesive. Thus a maximal pk-Clique community satisfies four desirable inner properties, i.e., society, cohesiveness, connectivity, and maximum.

The personalized top-n influential community search problem is very different from the most influential community search problem [22] and the influence maximization problem [6, 35]. The most influential community concerns the influence at the community level over entire network, and the influence maximization problem aims to select nodes as seeds in social networks such that the number of other nodes affected by the seed nodes is maximized. However, in the personalized top-n influential community, we are interested in the communities around the user which meets structural constraints, i.e., densely and cohesively connected. In other words, the personalized top-n influential community concerns the influence at the community level rather than the individual level, and the communities around a user rather than over entire network.

To find top-n influential communities efficiently, we developed a pruning and filter based search algorithm with the support of an auxiliary data structure. With the help of a priority queue, the algorithm keeps track of currently found top-n influential community. With the upper bound of the aggregated influence of maximal cliques in a subgraph, it performs pruning and filtering in the search space. We also developed a diversify method to retrieve the diversified results. A tree-based index, called D-Tree, is deliberately designed to support efficient diversified community search in one pass. And we have developed algorithms to efficiently find the top-n influential communities with the support of D-Tree. The contributions of our work can be summarized as follows:

- We propose a novel cohesive criterion to define an explicit community model in evolving and dynamic social networks;
- The method to identify the search space that contains all communities surrounding a user is proposed. Then we propose an exactly search algorithm that reduce time complexity by more than two times, while preserving the robustness of the search approaches;
- To conduct diversified top-n search in one pass, a diversify method is introduced. We design a tree-based index (D-Tree) to effectively maintain the top-n communities currently found in social networks and aid in diversifying the results;
- The experimental results of four real datasets demonstrate the correctness of our algorithms and show that the proposed algorithms significantly outperform the baseline algorithm.

The remainder of this paper is organized as follows. First, we introduce the related work in Section 2. Then we propose and formulate the personalized top-n influential community search problem in large social networks in Section 3 and present a baseline solution of the problem in Section 4. Improved algorithms are proposed to find top-n influential community with pruning and filter approach in Section 5. In Section 6, we discuss a diversify algorithm which conducts diversified search. The experimental results are analyzed in Section 7. Finally, this work is concluded in Section 8.

2 Related work

The detection of community, which is defined as natural divisions of network nodes into densely connected subgroups [25], has been widely studied in biological networks and social networks. Newman et al. [25] described a new class of algorithms for performing network clustering using a divisive technique. Ruan et al. [26] presented an efficient spectral algorithm for modularity optimization. The algorithm adopts a recursive strategy to partition networks while optimizing modularity. Instead of using a partitioning approach, Satuluri et al. [27] adopted a clustering approach and proposed a multi-level algorithm for graph clustering using flows. The graph is first successively coarsened to a manageable size, then refined, finally the algorithm is run to convergence and the high-flow regions are clustered together.

Concepts relating to graph properties like k-clique, k-core and so on have been extensively studied in random graphs. Recently, implicit community models k-core [23] and kr-Clique [22] have been proposed to discover communities. The works of Li et al. [23] focus on k-core decomposition in an evolving graph. They found that when maintaining the core number for every node in a dynamic graph, only certain nodes need to update their core numbers when the graph is changed by inserting/deleting an edge. The concept of maximal kr-Clique community [22] was introduced to search for the most influential community. A maximal kr-Clique community contains at least k nodes and any two nodes in the community can be reached in at most r hops via the shortest path in the subgraph. Huang et al. [16] propose a community model based on the k-truss concept and a compact index structure which supports the efficient search of k-truss communities with a linear cost with respect to the community size.

The online community search, which finds communities around a query vertex online, recently attracted a lot of attentions ([8, 16, 18] and [12]). Cui et al. [8] has proposed a novel approach and several algorithms for online overlapping community searches. They define a new model named α-adjacency-γ-quasi-k-clique and also provided theoretical guidelines for tuning the model. Fang et al. [11] propose attributed community query, which returns an attributed community for an attributed graph. An attributed community satisfies both structure cohesiveness and keyword cohesiveness. McAuly et al. [24] define a novel machine learning task of identifying users' social circles which uses a model combining network structure as well as user profile information. Chen et al. [4] propose co-located community search. They developed an efficient exact algorithm with several pruning techniques and an approximation algorithm with adjustable accuracy guarantees. Fei et al. [12] propose the top-k influential communities on-line problem. They introduce an instance-optimal algorithm LocalSearch whose time complexity is linearly proportional to the size of the smallest subgraph.

In considering models for the spread of influence through a social network, a dynamic cascade model called the Independent Cascade (IC) model, was first investigated by Goldenberg et al. [14]. Granovetter et al. [15] were among the first to propose another Linear Threshold (LT) model that was based on the use of node-specific thresholds to capture such a spread process. Kempe et al. [19] discussed maximizing the spread of influence problem with these models in social network analysis. Based on these works, there appears to be a lots of proposals focusing on the spread of influence problem, for example, [5, 21], and

[20]. Most of these algorithms aim to select nodes as seeds in social networks such that the number of other nodes affected by the seed nodes will be maximized.

Maximal clique enumeration is a fundamental graph problem and has been extensively studied. Most solutions for maximal clique enumeration (e.g., [2] and [1]) are backtracking search. Eppstein et al. [10] show that there exists a nearly-optimal fixed-parameter tractable algorithm for enumerating all maximal cliques in a sparse graph. A polynomial delay algorithm which enumerates all maximal cliques is given in [3]. Cheng et al. [7] propose a general framework for designing external-memory algorithms for maximal clique enumeration in large graphs. Matthew et al. [28] propose a parallel and memory-efficient algorithm for distributed high performance computing architectures. Wang et al. [31] propose an algorithm which providing a concise and complete summary of the set of maximal cliques. The diversified top-k clique search problem is studied in [33], which is to find top-k maximal cliques that can cover most number of nodes in the graph.

3 Preliminaries

In this section, we first introduce the necessary background on influence modeling, and then present our definition of community for influential community search. Finally we present the formal definition of the personalized top-n influential community search problem. The notations in this paper are shown in Table 1.

3.1 Influence modeling in a social network

In this paper, a social network is modeled as a directed graph. Consider a directed graph $G = (V, E)$ with an edge labeled $w : E \rightarrow (0, 1]$, where V is the set of vertices representing users of a social network and E is the set of edges connecting vertices representing user-to-user connections. For every edge $(u, v) \in E$, weight $w(u, v)$ denotes the propagation probability of the edge, which is the probability that v is influenced by u through the edge (u, v).

Table 1 Notations

Symbol	Description
$G(V, E)$	A graph, node set V, edge set E
$w(u, v)$	Propagation probability edge (u, v)
P_{uv}	A path between u and v
p	Threshold of maximum influence probability of edge (u, v)
MIP	Maximum Influence Path
s	A querying node
C_{pk}, C	A pk-Clique
$\mathbb{C}$	A collection of pk-Cliques
$Pr()$	Aggregated influence of a node set
$\Gamma(v)$	The adjacent nodes of v
ε	Threshold to filter out vertices with little influence
V_{in}	Nodes with influence $\geq \varepsilon$
V_{out}	Nodes with influence $< \varepsilon$

To model the influence process, we adopt the widely-accepted IC (Independent Cascade) model [19] in this paper. Based on work in complex systems theory, the IC model simulates aggregate consequences concerning local interaction between individual users of a social network. This diffusion model is first investigated in the context of marketing by Goldenberg et al. [14]. It consists of a framework in which each user's behavior is dictated by a predefined propagation probability and is a function of the state of other users with whom he interacts. When it starts with an initial active node u in step t, and is followed an activation process which unfolds in discrete steps. That is, in step t+1, a currently inactive neighbor v is affected by user u, in one period, change his state from "inactive" to "active" at probability $w(u, v)$. It is worth noting that no matter u succeeds in transforming its neighbors' state or not, it cannot make any further attempts in subsequent period. Again, the process continues until no more activations are possible in the network.

For a path $P = \langle v_0, v_1, ..., v_k\rangle$ in G, we define the propagation probability of the path as the product of the weights of its constituent edges.

$$w(P) = \prod_{i=0}^{k-1} w(v_i, v_{i+1}) \tag{1}$$

There may be more than one path between one vertex and the other, and different paths may have different propagation probabilities. We use the Maximum Influence Path (MIP) to approximate the real influence from one vertex to another within the social network. A maximum influence path from vertex u to vertex v is defined as any path P with a maximal weight $w(P)$.

3.2 Problem formulation

After introduction of graph G, influence model and MIP, we first define pk-Clique community and aggregated influence of community, then present the definition of the problem with an example followed.

Definition 1 (pk-Clique Community)
Given a graph G, and two parameters p and k, a pk-Clique community is an induced subgraph $C_{pk} = (V_{pk}, E_{pk})$ of G. It is a complete graph that satisfies the following requirements:

- Any two vertices in C_{pk} can be reached via an MIP between them in the subgraph C_{pk}, and the weight of every MIP is greater than p. And there are at least k vertices in C_{pk}.
- A pk-Clique community is maximal if C_{pk} is not a subgraph of any other pk-Clique community.

The difference in this study from previous community models [16, 22, 23] is that, for the first time, we model the relationship between users in a community with an edge as a dynamic propagation probability. We argue this is an appropriate measurement of cohesiveness in a community. For k-core model, a community is the largest subgraph within which each node has at least k connections. Since each node in the community is only required to have k neighbors, such a community may be less cohesive. The k-truss is a model based on triangle which describe the stable relationship among three nodes. With edge connectivity constraints, the induced community is connected and cohesive, and fits for communities in very small size. For kr-Clique community, it requires any two nodes can be reached at most r hops via the shortest path. Obviously there exists different propagation probabilities between any two nodes in a social network. Connections labeled "weak ties" are different from the more stable and intimate "strong ties" which have greater propagation probabilities. By taking dynamic propagation probability into consideration, the maximal pk-Clique community overcomes the drawbacks of the conventional community models

To calculate the influence of a community, the problem is that a vertex can be influenced by different vertices through different paths. Many works [22] adopt a model known as an "expectation model" to simplify the paths by supposing a dependence between paths, and mostly the MIP is the path that is adopted. In this work we are mainly concerned with the relationship between two vertices. We always assume that the paths between a different pair of vertices are independent. That is, even if two paths that share a sub-path lead to the same destination, we also assume that there are virtually two independent paths from two sources to the destination. This approximation simplifies the calculation of the aggregated influence of a community.

Definition 2 (Aggregated Influence of Community)

Aggregated influence of a maximal pk-Clique community to a vertex s, is defined as the influence probability that s is influenced by any vertices in this community, and the weight of every maximum influence path from these vertices to vertex s is greater than ε. Denote aggregated influence as $Pr(v \mid V(C))$ by the IC model, it is calculated as

$$Pr(v \mid V(C)) = 1 - \prod_{v \in V(c)} (1 - w(P_{v \to s})) \tag{2}$$

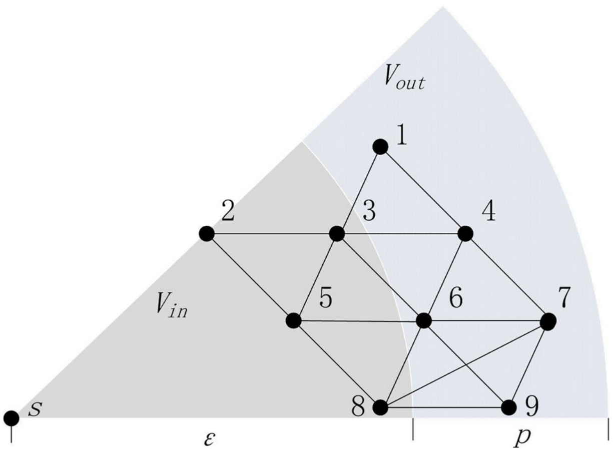


Figure 1 A Query Example of node s

Problem 1 (Top-n Influential Community Search)

Suppose $\mathbb{C}$ is the set of influential maximal pk-Clique communities around s. The top-n influential community search can be defined as a query to find n maximal pk-Cliques in $\mathbb{C}$, formally,

$$\text{Top-}n(s) = \langle C_1, C_2, ..., C_n \rangle \text{ where } Pr(C_1) > Pr(C_2) > ... > Pr(C_n) \text{ if } \mid \mathbb{C} \mid > n.$$

Example 3 (Top-3 maximal pk-Cliques in Figure 1) Consider a small social network as shown in Figure 1. It illustrates a subgraph surrounding a query node s. Suppose the vertices residing in the dark grey area have influence over s greater than ε (say, 0.2). To simplify the illustration, we set all their influence to s with their identity number divided by 10. That is the node 2 has influence to s with 0.2, node 3 is 0.3, and so on. When calculating aggregated influence of a clique, we ignore the vertices residing in the light grey area with an influence of less than ε. But when we check whether a group of vertices is a pk-Clique, these vertices make sense, since a maximal pk-Clique may include the vertices where the influence over s is less than ε. After having completed the search, we present all maximal pk-Cliques and their aggregated influence in Table 2. We identify clique {5,6,8} as top-1, {6,8,9,7} as top 2, {2,3,5} as the top-3 result.

Being an NP-hard problem, the personalized top-n influential community search is a task with high challenge. So we need an algorithm try to optimize the searching process with some greedy strategy. In other words, it should exploit some specific knowledge of the problem, whose purpose is to make the result top-$n(s)$ subject to particular constraints, with the aim of (1) reducing the algorithm search space or (2)possibly avoiding that the greedy generation of cliques not belonging to final results.

4 Basic solution

4.1 Search space

In contrast to community detection that aims to find all communities in a graph, personalized search of top-n influential communities only considers communities “near” a given vertex in the social graph. We take the word “near” to mean a vertex that can be reached by a specific community with a probability above a fixed threshold. So we only examine communities that include at least one vertex such that the MIP between this vertex and the query node is above the threshold ε.

Table 2 Maximal pk-Cliques (Example 3)

Clique(k=3)	Pr
{5,6,8}	0.9
{6,8,9,7}	0.8
{2,3,5}	0.72
{3,5,6}	0.65
{3,4,6},{1,3,4}	0.3
{6,4,7}	0

We adopt the single-source shortest-path algorithms such as Dijkstra algorithm to find all vertices around a query node where the weight of their MIPs is above ε. A modified version of Dijkstra algorithm is used , the algorithm will stop at a vertex when the propagation probability of the MIP between this vertex and the query node is less than ε.

We denote the vertices where the propagation probability to a query node is above ε as V_{in}. From previous discussions we are also aware that a maximal pk-Clique community C, said to be influential to a query node, must include at least one vertex that is in V_{in}. If we denote this vertex as v, to determine whether C is a maximal pk-Clique, we must examine every vertex where the propagation probability of the MIP between these vertices to v is greater than p.

We derive the following lemma.

Lemma 1 *(The minimum propagation probability of an MIP between a vertex in a maximal pk-Clique and the query node s.)*

Given a graph $G = (V, E)$ with influence function $w : E \to (0, 1]$, probability p, ε, and a query node s, for any vertex u, if the MIP between vertex u and node s is less than $p\varepsilon$, u will not be included by any maximal pk-Clique that has influence on s.

Algorithm 1 Search space.

```
Require:
        A graph $G = (V, E)$, vertex $s$ and probability $\varepsilon$, $p$
Ensure:
        A graph $G' = (V', E')$
 1: $V\_in = \{v \mid v \in V \,\&\, w(P_{vs}) > \varepsilon\}$ ; Set influence attribute of nodes with $w(P_{vs})$ ;
 2: $V\_out = \{v \mid v \in V \,\&\, \varepsilon > w(P_{vs}) > p\varepsilon\}$ ; Set influence attribute of nodes with 0;
 3: Copy $G' = (V', E')$ from $G$;
 4: for each edge $(u, v) \in E'$ do
 5:     if $w(u, v) < p$ then
 6:         Delete $(u, v)$ from $G'$ ;
 7: for each vertex $u \in V'$ do
 8:     for each vertex $v \in \{v \mid v \in V' \,\&\, w(P_{uv}) > p\}$ do
 9:         if $(u, v) \notin E'$ then
10:             Add $(u, v)$ to $G'$ ;
11: return $G'$;
```

Proof Suppose u is a vertex with $w(P_{us}) < p\varepsilon$. And $P_{us} = \langle u, v_1, v_2, ..., v_k, s\rangle$ is a maximal influential path from u to s. There will be a node $v_i (1 \le i \le k)$ that is the last node along path P with $w(P_{v_i s}) \ge \varepsilon$. In other words, node v_i is the last node in set V_{in} along path P.

Suppose node v_i is included in a maximal pk-Clique denoted by C and path P can be decomposed into P_{uv_i} and $P_{v_i s}$. We know that sub-paths of a maximal influential path are maximum influence paths. Then, paths P_{uv_i} and $P_{v_i s}$ are maximum influence paths. Because with $w(P_{v_i s}) \ge \varepsilon$, we have $w(P_{uv_i}) < p$. According to Definition 1, vertex u will not be included in C. □

With Lemma 1 and Definition 1, we propose Algorithm 1, that returns a candidate subgraph G', including all vertices around a query node s with influence to s greater than $p\varepsilon$.

Algorithm 1 obtains all the vertices consisting of V_{in} and V_{out}, and any edge between these vertices remains unchanged in graph G'. Since we use a modified version of Dijkstra's algorithm to retrieve influential vertices at lines 1-2, the time complexity of identifying V_{in} and V_{out} is $O(\mid E' \mid lg \mid V' \mid)$.

4.2 Basic algorithm

In this and the following sections we will use the terms maximal *pk*-Clique and maximal clique interchangeably. Using social networks related to a query node s, the first step of a top-n Influential Community Search (ICS) is to identify the subgraph, including all candidate cliques, by executing Algorithm 1. Then the ICS enumerates all maximal cliques in this area. Among all these returned maximal cliques, we filter out those cliques whose size is less than k, calculate the integrated influence of each community, and then push it into a heap. When all the maximal cliques have been put in the heap, the algorithm selects top-n maximal cliques in the heap and returns the result list.

Algorithm 2 Basic Top-n ICS.

```
Require:
        A graph $G = (V, E)$, parameter $n, k, l$
Ensure:
        A sorted list of cliques
 1: Run Algorithm 1;
 2: $heap = \emptyset$;   $list = \emptyset$;
 3: $\mathbb{C} \leftarrow$ MCE($V_{in} \cup V_{out}, \emptyset, \emptyset$);
 4: for each clique $C \in \mathbb{C}$ do
 5:     if $|C| \geq k$ then
 6:         Push($heap, C$);
 7: for $i = 1$ to $n$ do
 8:     $C$ = Pop($heap$);
 9:     $list.append(C)$;
10: return $list$;
```

Algorithm 3 MCE.

```
Require:
        Vertex set $P, X, C$
Ensure:
        maximal cliques
 1: if $P \cup X == \emptyset$ then
 2:     Output $C$ as a maximal clique;;
 3:     return;
 4: Choose a pivot $u \in P \cup X$;
 5: for each vertex $v \in P \setminus \Gamma(u)$ do
 6:     MCE($P \cap \Gamma(v), X \cap \Gamma(v), C \cup \{v\}$);
 7:     $P = P \setminus \{v\}$;
 8:     $X = X \cup \{v\}$;
```

The procedure of the top-n influential community search is presented in Algorithm 2. Algorithm 2 calls the procedure, Maximal Cliques Enumerate (MCE), to generate all the maximal cliques in a graph. MCE is an implementation of maximal clique enumeration method proposed by Bron-Kerbosch [2]. Its performance is discussed in [30]. The basic idea of MCE is to recursively backtrack to add a vertex from the set P which consists candidate vertices, to extend the current clique C. MCE tends to produce the larger cliques first and to generate sequentially cliques having intersection. As Algorithm 3 indicates, it takes three sets as input. (1) The set P is the candidate set to be extended by a new vertex along a branch of the backtracking tree. (2) The set X is a set of vertices that have already been processed by the algorithm. (3) The set C is the set of all vertices that have already served as an extension of the present configuration of P. That is, C is a clique found along the branch of the backtracking tree. The set of vertices $P \cup X$ serves as search space. Once $P \cup X$ becomes empty, C cannot be further extend. At this point, we output C as a maximal clique. A vertex v from the search space is selected to grow the current clique C. To grow clique C, only the vertices in $P \cap \Gamma(v)$ and $X \cap \Gamma(v)$ can be candidates for expanding to find new larger cliques. After we finish searching in $(P \cup X) \cap \Gamma(v)$, vertex v is excluded from P and placed in X.

Algorithm 3 takes G' as input, where G' is originally obtained from Algorithm 1. Before being fed to Algorithm 2, it is transformed into an undirected graph. This searching process of MCE can be represented by a search forest, or the collection of search trees. The set of roots of the forest is exactly the same as V' of graph G'. For each $v \in V'$, all the vertices in $V' \cap \Gamma(v)$ are children of v. Thus, vertices along a path from the root v to any leaf vertex of the tree constitutes a maximal clique.

After collecting all maximal cliques at line 3, Algorithm 2 pushes every maximal clique into a data structure *heap*. As its name implies, *heap* uses the heap to manage maximal cliques during the insertion. The sort key is the aggregated influence of each clique. Lines 5-6 finish this operation. After we have the ordered cliques, the next step is to output top-n results. Lines 7-9 pop cliques from *heap* and appended the cliques to the result list.

Algorithm 2 achieves the worst-case time complexity of $O(3^{|V|/3})$ for a $|V|$-vertex graph [30]. By using d-degenerate order in the first-round iteration, the complexity is reduced to $O(|V|3^{d/3})$ [10].

5 Efficient influential community search

Algorithm 2 is a functionally correct procedure, but we are unlikely to be satisfied with its performance, particularly with the large social networks that now exist. Since maximal clique searching is a combinatorial optimization problem, it's well know that domain Constraint Programming (CP) has been recognized as a powerful tool for modelling and solving combinatorial optimization problems [13]. CP aims at removing from variable domains combinations of values which cannot appear in any desired solution. And when dealing with optimization problems, pruning of search trees can be performed on the basis of currently known costs or best results. Filtering can be used to remove combination of vertices which cannot lead to solutions. Following this line, in this section, we introduce pruning and filtering approach in influential community search.

5.1 A pruning approach

Domain CP usually implements a branch and bound pruning algorithm to search an optimal solution based on some domain specific knowledge. The idea is to solve a domain specific problems, each time leading to successively better results (i.e., a feasible result is get if it exists). In particular, when a feasible result r^* is found (whose cost or earning is $f(r^*)$), a constraint $f(x) < f(r^*)$ is introduced to each subtree in the remaining search tree. The purpose of the introduced constraint, called upper bounding constraint, is to get rid of portions of the search space which cannot lead to better results than the best one get so far. The problems in this branch and bound pruning approach are: (1) what can be used as the the upper bounding constraint; (2) how about the monotonicity of the cost function.

The purpose of the search is to find top-n influential pk-Clique community. Algorithm 1 has identified the search space that ensures any two vertices in a found communities can be reached via an MIP with the weight greater than p. We also must ensure there are at least k vertices in any returned communities and only need to return the top-n results. To this end, we need to make sure that any candidate set of vertices used to grow a maximal pk-Clique has at least k vertices and its aggregated influence is at least greater than currently achieved the n-th maximal clique.

We consider the algorithm MCE for generating all the maximal cliques of a given graph $G = (V, E)(V \neq \emptyset)$. Suppose variable C is a set of vertices that constitutes a clique found up to this time. MCE begins by letting C be an empty set, and expands C step by step by applying a recursive procedure to V. It will induce subgraphs to achieve larger and larger cliques until they reach maximal ones.

Let $C = \{v_1, v_2, ..., v_d\}$ be a clique found at some stage, and consider the set of vertices

$$S = V \cap \Gamma(v_1) \cap \Gamma(v_2) \cap \Gamma(v_3) \cap ... \cap \Gamma(v_d) \tag{3}$$

where $S = V$ and $C = \emptyset$ at the initial stage. Now, consider the smaller subgraphs $G = (S_v, E_v)$ that are induced by new sets of vertices

$$S_v = S \cap \Gamma(v) \tag{4}$$

and at this stage we have C_v. We also can denote the set of S_v with remaining candidates and finished vertices for expansion by

$$S_v = P_v \cup X_v \tag{5}$$

So at certain stage of MCE, $C_v \cup v \cup S_v$ is used to grow a possible maximal clique. We first propose Lemma 2, which constrain the lower bound of vertices in a pk-Clique.

Lemma 2 *(The lower bound of vertices during MCE.)*
On each recursive stage of MCE, there must exist at least k vertices in $C_v \cup v \cup S_v$ to grow a pk-Clique.

Proof It's established from the definition of pk-Clique. □

Since we are exploring the possibility of taking the influence of the n-th maximal clique as the lower bound of to-be-found communities. Is it possible to derive an upper bound of aggregated influence of to-be-found communities in advance. And with this bound, we can prune unnecessary search branches in the graph. To obtain such an upper bound, we must prove the monotonicity of a vertex set's aggregated influence. Recalling Definition 2, the aggregated influence of the community is defined as the probability that s is influenced by any vertex in a maximal pk-Clique community. Hence we have Definition 3, and then Lemma 3.

Definition 3 (Aggregated Influence of vertex set)
The aggregated influence of a set of vertices to a vertex s, is defined as the probability that s is influenced by any vertex in this set. The calculation is the same with that in Definition 2.

We then propose Lemma 3.

Lemma 3 *(Monotonicity of the aggregated influence during MCE.)*
On each recursive stage of MCE, the aggregated influence of $C_v \cup v \cup S_v$ is monotone decreasing.

Proof Suppose we have C_{v_i}, v_i and S_{v_i} at stage i, and $C_{v_{i+1}}$, v_{i+1}, $S_{v_{i+1}}$ at next stage, where $0 < i < d$.
It's obvious that

$$C_{v_{i+1}} = C_{v_i} \cup \{v_i\} \tag{6}$$

$$S_{v_{i+1}} = S_{v_i} \cap \Gamma(v_{i+1}) \tag{7}$$

and

$$v_{i+1} \in S_{v_i} \tag{8}$$

Let $SC_i = C_{v_i} \cup \{v_i\} \cup S_{v_i}$ and $SC_{i+1} = C_{v_{i+1}} \cup \{v_{i+1}\} \cup S_{v_{i+1}}$.
We have

$$SC_i \supseteq SC_{i+1} \tag{9}$$

According to equation 2, we have

$$\begin{aligned} & Pr(u \mid u \in SC_i) - Pr(u \mid u \in SC_{i+1}) \\ & = 1 - \prod_{u \in SC_i} (1 - w(P_{u \to s})) - (1 - \prod_{u \in SC_{i+1}} (1 - w(P_{u \to s}))) \\ & = \prod_{u \in SC_{i+1}} (1 - w(P_{u \to s}))(1 - \prod_{u \in SC_i \setminus SC_{i+1}} (1 - w(P_{u \to s}))) \\ & > 0 \end{aligned} \tag{10}$$

Hence have the result. □

With the above lemmas, we now know the aggregated influence of a vertex set is greater than any maximal clique residing in it. We have the upper bound of the aggregated influence of maximal cliques in a subgraph. This upper bound will help us make the right decision before exploring the branch of a search tree. And we also know how to compute this upper bound.

Now we introduce Algorithm 4 which prunes unnecessary branches from a search tree in the forest.

Algorithm 4 Improved Top-n ICS.

```
Require:
        A graph $G = (V, E)$, parameter $n, k$
Ensure:
        A sorted list of cliques
 1: $q = []$;
 2: $list = \emptyset$;
 3: MCP($V_{in} \cup V_{out}, \emptyset, \emptyset, k$);
 4: for $i$ = 1 to $n$ do
 5:     $C$ = Dequeue($q$);
 6:     $list.append(C)$;
 7: return $list$;
```

Algorithm 5 MCP(P, X, C.).

```
Require:
        Vertex set $P, X, C$
        Global varible $q, n, k$
Ensure:
        maximal cliques
 1: if $P \cup X == \emptyset$ then
 2:     Insert $C$ into $q$;
 3:     return;
 4: Choose a pivot $u \in P \cup X$;
 5: for each vertex $v \in P \setminus \Gamma(u)$ do
 6:     $T = C \cup \{v\} \cup ((P \cup X) \cap \Gamma(v))$;
 7:     if $Pr(T) \leq Pr(q[n])$ or $|T| < k$ then
 8: return
 9:     MCP($P \cap \Gamma(v), X \cap \Gamma(v), C \cup \{v\}$);
10:     $P = P \setminus \{v\}$;
11:     $X = X \cup \{v\}$;
```

Algorithm 4, Improved Top-n ICS, calls Algorithm 5, MCP (Maximal Cliques with Pruning), to complete the top-n influential community search. It use an efficient global priority queue to manage the found maximal cliques and returns a list of top-n influential community.

Algorithm 5 MCP takes more inputs than Algorithm 3 MCE. It takes global priority queue q, n and k to conduct branch and bound pruning.

The second main difference between MCP and MCE is lines 5-7. Vertex set T denotes a union of vertex sets, including C, $\{v\}$, and $(P \cup X) \cap \Gamma(v)$. Set C includes the vertices that are part of maximal cliques to be found. $\{v\}$ is currently an expanding vertex. And $(P \cup X) \cap \Gamma(v)$ includes vertices to be extended. According to Lemma 3, the aggregate influence of set T is the upper bound of any pk-Cliques to be retrieved. Thus, if the aggregate influence of set T is less than the minimum aggregate influential pk-Cliques in the queue, which is clique

$q[n]$, there is no need to undergo further searches along this branch. For the same reason, if the size of T is less than k, there is no need to conduct further searches either. This is the pruning based top-n influential community search. When a maximal clique is found, it is also important to notice that the size of any maximal clique MCE derived is not always greater than k. Thus we also need to check if a derived maximal cliques is less than k before it is inserted into q. This operation is executed with line 2. At the end of MCP, we have a queue with n maximal pk-Cliques.

Now we analyze its time complexity. In the best case, MCP finds top-n maximal pk-Cliques first. Thus the time complexity is $O(n)$. But in the worst case, MCP finds top-n maximal pk-Cliques at the last moment. Then the time complexity in this case is same with MCE, and is $O(3^{|V|/3})$ for a $|V|$-vertex graph.

5.2 Filtering unqualified vertex

Another main principle of CP is that every constraint is associated with a domain reduction algorithm or filtering algorithm that removes some vertices which are inconsistent with the constraint. The deletions of vertices in the search space involve a propagation mechanism that utilizes the filtering algorithm of the constraint until no more modification occurs.

In this section we use a constraint based on the degree. This constraint is imposed at the beginning of the algorithm.

Next we introduce a lemma first and then the filtering algorithm.

Lemma 4 *(Degree constraint of a vertex in a pk-Clique.)*
A vertex with degree less than $k-1$ cannot be a member of any maximal pk-Clique.

Proof It's established from the definition of pk-Clique. □

The filtering algorithm uses degree to remove all vertices whose degree is too small to extend to a possible clique of size greater than k. Many core decomposition algorithms determine a vertex's class in a bottom-up manner. That is, it sorts vertices in order of their degree and get rid of vertices whose degree is less then k. We adopt such a bottom-up approach to remove all vertices, so they will not be included in any maximal pk-Clique.

In implementation of Algorithm 6, it's convenient to organize vertices currently under considering into min-heap, and take a vertex at the root to determine and remove at each time. With heap sort, Algorithm 6 takes time $O(|V|lg|V|)$.

5.3 Heuristic search

According to Definition 2, the aggregated influence is calculated as the union of the influence of every vertex in a clique. If the probability distribution of each node's influence in V_{in} is uniform, then the more nodes that reside in V_{in} for a clique, the bigger the aggregated influence it will acquire. This leads to following observation.

Observation: It is generally expected that a vertex $u \in V$ such that $V = V_{in} \cup V_{out}$ and $u = max\{|V_{in} \cap \Gamma(u)|\}$ has a high probability of being included in an influential maximum clique.

With above observation, we finally propose Algorithm 6, an improved version MCP, which incorporates heuristic search and vertex filtering. To minimize $P \setminus \Gamma(u)$, MCP chooses a pivot $u \in P \cup X$ that maximizes $|P \cap \Gamma(u)|$ at line 3 of Algorithm 5. In contrast to MCP, This new version MCP chooses a pivot that maximizes $|V_{in} \cap P \cap \Gamma(u)|$. That is, expending a subtree that has a high probability of leading to influential maximum cliques becomes a priority.

Algorithm 6 VertexFiltering.

```
Require:
        A graph $G = (V, E)$, Clique $C$
Ensure:
        A graph $G' = (V', E')$
 1: $G' = G$;
 2: Sort vertices in $G$ in ascending order of their degree;
 3: while ($G' \neq \emptyset$) do
 4:     $d$ = the minimum vertex degree in $G'$;
 5:     if $d < k$ then
 6:         for each $v \in \{v \mid v \in V' \ \& \ v.degree \leq d\}$ do
 7:             Delete $v$ and all edges incident to $v$ from $G'$;
 8:             Re-sort the remaining vertices in $G'$;
 9:     else
10:         Break;
11: return $G'$;
```

The main program of heuristic search based top-*n* ICS is almost the same as Algorithm 4, except it maintains an additional list which keeps the sets of all vertices that are in V_{in} and adjacent to v in $G' = (V', E')$. Note that the rather time-consuming calculation at this step is carried out only at the beginning of the main program and not in HMCP. Therefore, the total time to select the pivot is very small.

Algorithm 7 MCP(P, X, C).

```
Require:
        Vertex set $P, X, C$
        Global varible $q, n, k$
Ensure:
        Maximal cliques
 1: if $P \cup X == \emptyset$ then
 2:     Insert $C$ into $q$;
 3:     return;
 4: Choose a pivot $u \in P \cup X$
                where $u = max\{|V_{in} \cap P \cap \Gamma(u)|\}$
 5: Same with Algorithm 5 lines 5-11;
```

Example 4 (An execution of a heuristic search) With the settings left the same as in Example 3, it is easy to find a heuristic based search that will perform the outermost recursive calls in the order of {5,7,1}, instead of {6,2,1} in MCP.

5.4 Analysis

The improvement of basic top-n influential community search focuses on filtering of unqualified vertices and brunch-bound pruning of search trees. We prove the correctness of such an approach in Theorem 1.

Theorem 1 *(Correctness of algorithm Improved Top-n ICS).*
Given a graph $G = (V, E)(V \neq \emptyset)$ and a query node s, the algorithm Improved Top-n ICS generates top-n maximal pk-cliques without duplication.

Proof It has been proved in [30], that MCE generates all, and only, maximal cliques that contain all vertices in C, some vertices in P, and no vertices in X, without duplication. MCP follows MCE, and it removes some vertices in the searching space according to Lemma 4, and skips over some branches on the search tree that do not lead to top-n maximal cliques according to Lemma 2, 3. Thus MCP returns the top-n maximal cliques with input vertex sets P, X, C as well. Improved Top-n ICS call procedure MCP to finish its search of top-n maximal cliques, hence correctness is insured by MCP.

□

6 Diversified top-*n* search

The Improved Top-n ICS is an exact algorithm. The cohesiveness measurement used in pk-Clique is reasonable. The introduction of pk-Clique will add more new edges to a network, thus leading to an increased number and size of cliques. Though it improves the efficiency of enumerating maximal cliques in the graph, it still suffers from exploring a huge search space and a significant level of redundancy, like many existing maximal clique enumeration algorithms [31].

An exact algorithm of maximal cliques enumerating usually returns a significant level of redundancy as many maximal cliques share a large portion of common vertices. Thus, it is not very useful to report similar top-n maximal cliques since they bring about a large amount of duplicate information. Thus an approximate or diversified top-n cliques is meaningful in some circumstances. For example, the purpose of such diversification is to return the results that can best cover different possible categorizations of the query [34].

In order to compute diversified top-n pk-Cliques, a straightforward solution is to enumerate all top-n' cliques first, where n' must be sufficiently great, and then pick out n diversified cliques with a specified threshold. However, since it requires two passes over created cliques, such a solution falls short to handle large amount cliques in worst-case, and might be a costly operation.

In this section, we introduce an one-pass diversify algorithm for top-n ICS. First we introduce the notion of l-diversity, which is used to ensure that every maximal clique in the result is different with each other. Then we propose a deterministic algorithm based on l-diversity. The algorithm utilizes properties of backtracking MCP algorithms for further redundancy identification and search space pruning. To compute l-diversity, we use an tree data structure to index current top-n maximal cliques to match with to be generated ones.

6.1 *l*-diversity

We first introduce the notion of l-diversity that measures how well each maximal clique is diversified with existing cliques.

Definition 4 (l-diversity)
Given two maximal cliques C_i, C_j in graph G and a parameter $l(0 \leq l \leq 1)$. We define the diversity of two maximal cliques C_i and C_j as

$$D(C_i, C_j) = 1 - \frac{\mid C_i \cap C_j \mid}{\mid C_i \cup C_j \mid} \tag{11}$$

If $D(C_i, C_j) \geq l$, then cliques C_i and C_j are l-diversified, otherwise cliques C_i and C_j are l-similar.

Because the searching process of MCP follows a depth-first order. Any two cliques' outputs in order tend to be similar if they are near in the output sequence. Thus it's possible to report cliques while avoiding redundant with previous ones and further pruning the search space. Specifically, such pruning is performed when the MCP process is entering a new stage of recursion.

Let C' be one of the maximal clique that previously added to top-n cliques Q. At a stage of MCP(P, X, C), let $\bar{d}$ be an upper bound depth of the to be created search tree. Then, the size of each clique C_p to be generated from P is at most $|C| + \bar{d}$.

To get the lower bound of $|C_p \cap C'|$, let $Y = P \setminus C'$, and $t = |C_p \setminus C|$. It's easy to know that part of the t vertices come from Y and the rest are from $P \cap C'$. And we also assume at most $\bar{t_y}$ vertices out of t come from Y. Then since C_p is extended from C, we have $|C_p \cap C'| \geq |C \cap C'| + max\{t - \bar{t_y}, 0\}$, where $1 \leq t \leq \bar{d}$. Similarly we also have the upper bound that $|C_p \cup C'| \leq |C \cup C'| + \bar{t_y}$.

We focus on whether it's necessary to enter a new stage of MCP. We need to check if a maximal clique C currently under growing is similar with existing cliques in the queue while considering its possible growth in P at the same time. If it's similar with some existing cliques, then we stop to extend C in candidate set P. Otherwise, we enter the new stage of MCP. We define the maximal diversity of C as following equation.

$$\begin{aligned} D_c &= 1 - \max_{C' \in Q} \frac{|C \cap C'|}{|C \cup C'| + \bar{t_y}} \\ &\geq 1 - \frac{|C|}{|C| + \bar{t_y}} \end{aligned} \tag{12}$$

It's easy to notice from equation 12, that we don't need to check if a due-to-created clique is similar with existing ones. The inequation $D_c < l$ only will be satisfied by

$$\frac{\mid C \mid}{\mid C \mid + \bar{t_y}} \geq 1 - l \tag{13}$$

6.2 Diversified MCP

In this section, we introduce Algorithm 8 DiversifiedMCP as follows.

Algorithm 8 DiversifiedMCP(P, X, C, C').

```
Require:
        Vertex set $P$, $X$, $C$, and $C'$ used to indicate if it's a approximate search
        Global variable $q, n, k, l, Q$
Ensure:
        Maximal cliques
 1: if $P \cup X == \emptyset$ then
 2:     if $C' == \emptyset$ then
 3:         Insert $C$ into $q$ and $Q$;
 4:     else
 5:         Replace $C'$ with $C$ in $q$ and $Q$;
 6:     Update $Q$;
 7:     return;
 8: Choose a pivot $u \in P \cup X$
                where $u = max\{|V_{in} \cap P \cap \Gamma(u)|\}$
 9: for each vertex $v \in P \setminus \Gamma(u)$ do
10:     $T = C \cup \{v\} \cup ((P \cup X) \cap \Gamma(v))$;
11:     if $Pr(T) \leq Pr(q[n])$ or $|T| < k$ then
12: return
13:     if $C' == \emptyset$ and $\frac{|C|}{|C|+|P|} \geq 1 - l$ then
14:         Search for the most similar clique $C'$ in $Q$, where $D_c \leq l$;
15:     if $C'! = \emptyset$ then
16:         if $Pr(T) \geq Pr(C')$ then
17:             DiversifiedMCP($P \cap \Gamma(v)$, $X \cap \Gamma(v)$, $C \cup \{v\}$, $C'$);
18:         else
19: return
20:     else
21:         DiversifiedMCP($P \cap \Gamma(v)$, $X \cap \Gamma(v)$, $C \cup \{v\}$, $\emptyset$);
22:     $P = P \setminus \{v\}$;
23:     $X = X \cup \{v\}$;
```

The algorithm DiversifiedMCP takes one more parameter C' than MCP. C' is a clique existing in q which keeps track of currently known maximal cliques. In addition to C', the algorithm defines l as the threshold of diversity of the expected results. DiversifiedMCP also uses a maximal cliques set Q, which consists of the same cliques within q but organized as a red-black tree. We will introduce the organization of Q in more details later.

Each time upon DiversifiedMCP got a maximal clique C, whether it will be inserted into the q or replace an old clique C', which would depend on C'. If C' is empty, then it means exact maximal clique search. Otherwise, it is diversified search, which means C has a similar clique C' in q and the aggregated influence of C is greater than that of C'. In this case, C' in q needs to be replaced the latest found clique C.

Lines 13-14 in DiversifiedMCP indicate that the algorithm would lookup in Q to make sure if there is no cliques similar with C and diversity between them is less than l when entering a new recursion. Lines 15-17 show that if there is a such clique C' and the aggregated influence of the to be generated maximal clique might be greater than C', then the algorithm must conduct diversified search on the sub-tree. Otherwise it would abandon the search.

6.3 Diversified search tree

In Algorithm 8, when entering a new stage of extending clique C, C might be tested for similarity with the existing maximal cliques in the current Q. It is costly to compare C with every clique in Q. We devise the following novel method to process efficient similarity test for every to-be found maximal clique with the maximal cliques in Q.

Let low and $high$ be the smallest and greatest vertex id in a maximal clique C, respectively. We denote an interval$[low_c, high_c]$ as a clique. Then the range $[low_c, high_c]$ covers every vertices in C. Instead of comparing C with every $C \in Q$, we concern C when $[low_c, high_c] \cap [low_{c'}, high_{c'}]$ is nonempty. To locate such a C quickly, we augment a black-red tree to represent Q, named as Approximate Search Tree(AS-Tree), in which each node c contains an interval $c.int$ and the key of c is the smallest vertex id, $c.int.low$, of the interval. In addition to the intervals themselves, each node c contains a value $c.max$, which is the maximum id of any vertex id in cliques stored in the subtree rooted at c. Each node also contains clique C and its sequence number in q. Figure 2 shows how an AS-Tree represents a set of maximal cliques.

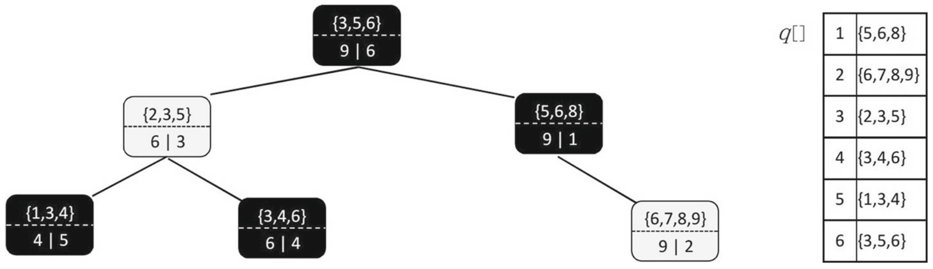
DS-Tree supports the following operations:

DS-Tree-INSERT(Q, C) creates a node c, which is assumed to contain the clique C, adds the node c to the tree Q.

DS-Tree-REPLACE(Q, C, C') replaces the node c, which is assumed to contain the clique C', with a created node which contains C in the tree Q.

DS-Tree-WALK(Q, C) returns a clique contained in a node, such that the similarity of this clique with C is maximal, or Ø if there is no such a node in the set.

The DS-Tree-INSERT(Q, C) operation is simple, since there is only one additional operation which is creating a node with input C involved, compared with general insertion in a red-black tree. The AS-Tree-REPLACE(Q, C, C') operation is like deletion in a red-black tree. The different between them is AS-Tree-REPLACE(Q, C, C') uses a new node to replace an old one instead of delete. It's obvious that insertion and replacement run in $O(lgn)$ time.



1	{5,6,8}
2	{6,7,8,9}
3	{2,3,5}
4	{3,4,6}
5	{1,3,4}
6	{3,5,6}

Figure 2 An diversified search tree and corresponding queue

The only new operation we add is DS-Tree-WALK(Q, C), which finds a node in tree Q, whose contained clique is the most similar with C. If there is no such a node in the tree, the procedure returns Ø.

Algorithm 9 DS-Tree-WALK(Q, C).

```
Require:
        Tree $Q$ and clique $C$
Ensure:
        Clique $C_{similar}$
1: if $Q$ == NULL then
2: return Ø;
3: Create node $c$ with $C$;
4: $C_{this} = Q.clique$;
5: $C_l$ = Ø;
6: $C_r$ = Ø;
7: if $Q.left.max \le c.int.low$ then
8:     $C_l$ = DS-Tree-WALK($Q.left$, $C$);
9: if $Q.int.low < c.int.high$ then
10:     $C_r$ = DS-Tree-WALK($Q.right$, $C$);
11: Set $C_{similar}$ as the most similar clique in $\{C_{this}, C_l, C_r\}$;
12: return $C_{similar}$;
```

As an example of a successful search, suppose we wish to find the most similar clique with clique {1,2,3} in the tree of Figure 2. We first create a node c with $c.low = 1$ and $c.high = 4$ which also contains C, then begin with the search from the root. Since the intersection of roots clique {3,5,6} and {1,2,3,4} is {3}, we have $C.this$ as {3}. And since $Q.left.max \le c.low$, we go left to the node {2,3,5}. As we can see, it will return c_l which is {2,3,5}. Because $Q.int.low \ge c.high$, we don't go right. We have $c.r = Ø$ at this point. From {{3},{2,3,5},Ø}, we finally return the clique {2,3,5}.

Lemma 5 *(Correctness of DS-Tree-WALK(Q, C).)*
DS-Tree-WALK(Q, C) returns the most similar cliques in tree Q for C.

Proof DS-Tree-WALK() follows divide-and-conquer approach and solve the problem recursively, at each level of the recursion it has three steps:

Divide the problem into three cases: I. The root is the result; II. If $[c.low, c.high]$ overlap with $(-\infty, Q.left.max]$, then check Q's left child; III. If $[c.low, c.high]$ overlap with $[Q.int.low, +\infty]$, then check Q's right child.

Conquer the subproblems recursively. If Q == NULL, just return Ø.

Combine the results from above three cases into the correct result and return.

After the recursion finished, it's obvious that it would return the most similar cliques in tree Q for C. □

6.4 Analysis

The algorithm DiversifiedMCP is invoked by Improved Top-n ICS to replace the exact algorithm MCP. The parameter q to help DiversifiedMCP track maximal cliques created. To ensure the correctness of DiversifiedMCP, we must guarantee maximal cliques in q is l-diversified. The following theorem shows its correctness.

Theorem 2 *(Correctness of algorithm DiversifiedMCP).*
Given P, X, C, C', and q, the algorithm ApproximateMCP(P, X, C, C') returns l-diversified maximal pk-cliques.

Proof The recursion of DiversifiedMCP() terminates when $P \cup X == \emptyset$ either when $C' == \emptyset$ or not. In the latter case, it would replace C' with latest created C, and is certainly correct to maintain all cliques in the result q are l-diversified. Therefore, we focus on the first case, in which the latest created C would be inserted into q.

We use the following invariant for the recursion:

If q contains maximal cliques, then those cliques are l-diversified.

We use this invariant as follows:

Initialization: Prior to the first recursion, q is empty, so that the invariant holds.

Maintenance: Each recursion of the DiversifiedMCP(), after searching for the most similar clique C' in Q, if return $\emptyset$, it shows there is no clique which similar with currently extending clique C. Then it would enter next stage of recursion. In each stage of recursion, if create a maximal clique, then it would be inserted into q. So this insertion maintains the invariant.

Termination: If the recursion terminates after traversing the entire search forest, then there is no more insertion at this point. Hence it is correct to return q.

Thus, the DiversifiedMCP procedure works correctly. □

7 Experimental study

In this section, we study the performance of the proposed algorithms over four real datasets. All the algorithms are implemented in Python2.7 and run on a CentOS server (Intel i7-7700 3.6GHz CPU with 4 cores and 32GB RAM).

Table 3 Statistical information of the datasets

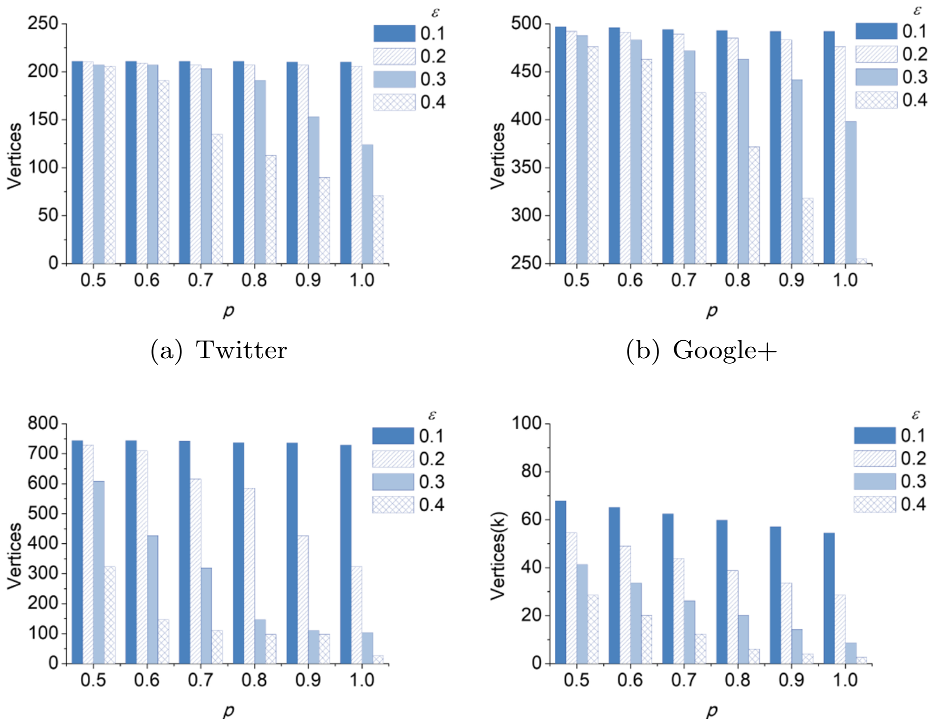
Dataset	Node	Edge	Diameter
Twitter	81306	1768149	7
Google+	107614	13673453	6
Facebook	4039	88234	8
Epinions	75879	508837	14

A. *Data sets*

We use four real social network datasets which are available at https://snap.stanford.edu/data/, as shown in Table 3. To verify performance of the proposed algorithm under different settings, we need a relatively large graph. So for each dataset, we focus on the largest subgraph. In each subgraph, we select one of the vertices with the most neighbors as the query nodes (i.e., vertex 2070 in Facebook, vertex 349932090 in Twitter, vertex 104***590 in Google+, and vertex 1 in Epinions). The degrees of chosen vertices are 12, 187, 138 and 839 in Facebook, Twitter, Google+ and Epinions, respectively. For those with no propagation probability associated with each edge in the original datasets, we set a random value which uniformly distributed in (0,1) to each edge as their propagation probabilities.

B. *Evaluation of different ε and p*

Figure 3 shows the number of nodes when we vary the value of p from 0.5 to 0.9 and the value of ε from 0.1 to 0.4. To demonstrate the performance of the proposed algorithm, experiments in this work focused on the scenario when $\varepsilon = 0.2$ and $p = 0.6$. When $p = 0.6$, the average length of paths from source to sink is 1.0 for Twitter, 1.96 for Google+, 1.91 for Facebook, and 1.99 for Epinions.



(a) Twitter (b) Google+

(c) Facebook (d) Epinions

Figure 3 Evaluation of different ε and p

Table 4 Efficiency over Twitter

k	5	15	25	35	45	55
HS	11.15	10.55	10.21	8.22	8.04	9.26
FS	10.51	10.34	9.95	8.20	8.16	9.31
PS	10.62	10.68	10.67	10.69	10.67	10.69
BS	131.12	131	130	132	130	129

Table 5 Efficiency over Google+

k	5	15	25	35	45	55
HS	0.83	0.81	0.51	0.17	0.05	–
FS	0.81	0.79	0.52	0.18	0.05	–
PS	0.82	0.79	0.78	0.75	0.70	–
BS	1.16	1.16	1.15	1.14	0.94	–

Table 6 Efficiency over Facebook

k	4	8	12	16	20	24
FS	2.94	2.89	2.79	2.70	2.50	2.28
HS	2.72	2.68	2.73	2.74	2.70	2.71
PS	2.68	2.72	2.76	2.73	2.71	2.96
BS	14.61	14.83	14.98	15.70	15.20	14.70

Table 7 Efficiency over Epinions

k	4	8	12	16	20	24
HS	9.19	9.04	9.00	8.98	9.01	8.63
FS	9.08	8.91	8.93	8.85	8.53	8.56
PS	9.08	8.89	9.08	8.77	8.80	8.58
BS	20.39	19.94	19.37	19.33	19.09	19.53

Table 8 Space cost over Twitter

k	5	15	25	35	45	55
HS	725	721	112	<10	<10	<10
FS	720	704	32	<10	<10	<10
PS	899	781	72	<10	<10	<10
BS	2.0E6	3.6E4	131	119	80	21

Table 9 Space cost over Google+

k	5	15	25	35	45	55
HS	1338	<10	<10	<10	<10	–
FS	716	<10	<10	<10	<10	–
PS	720	<10	<10	<10	<10	–
BS	9764	942	65	462	<10	–

Table 10 Space cost over Facebook

k	4	8	12	16	20	24
HS	643	643	<10	32	<10	–
FS	512	733	8	32	<10	<10
PS	741	749	20	<10	<10	<10
BS	2.0E5	1.9E5	6950	5693	5677	5673

Table 11 Space cost over Epinions

k	4	8	12	16	20	24
HS	8117	<10	<10	<10	<10	<10
FS	13947	102	<10	<10	<10	<10
PS	19357	<10	<10	<10	<10	<10
BS	63463	6733	<10	<10	<10	<10

Table 12 Scalability over Facebook

Vertices	97	147	616	744
HS	0.0027	0.005	4.61	1189.98
FS	0.0023	0.004	4.54	1179.98
PS	0.0023	0.005	4.73	1219.32
BS	0.0049	0.107	31.95	24384.70

Table 13 Scalability over Epinions

Vertices	4104	20151	43892	65169
HS	6.22	9.00	9.24	9.54
FS	6.19	8.86	9.11	9.27
PS	6.38	9.54	10.13	10.57
BS	14.83	20.32	20.73	21.22

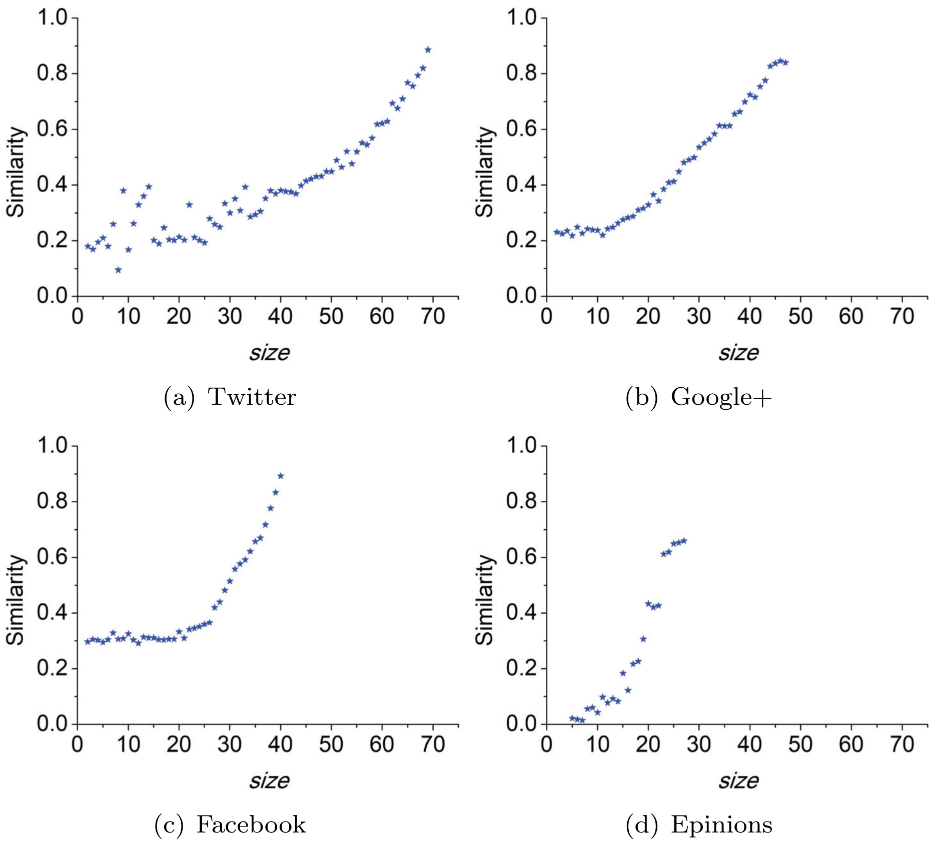


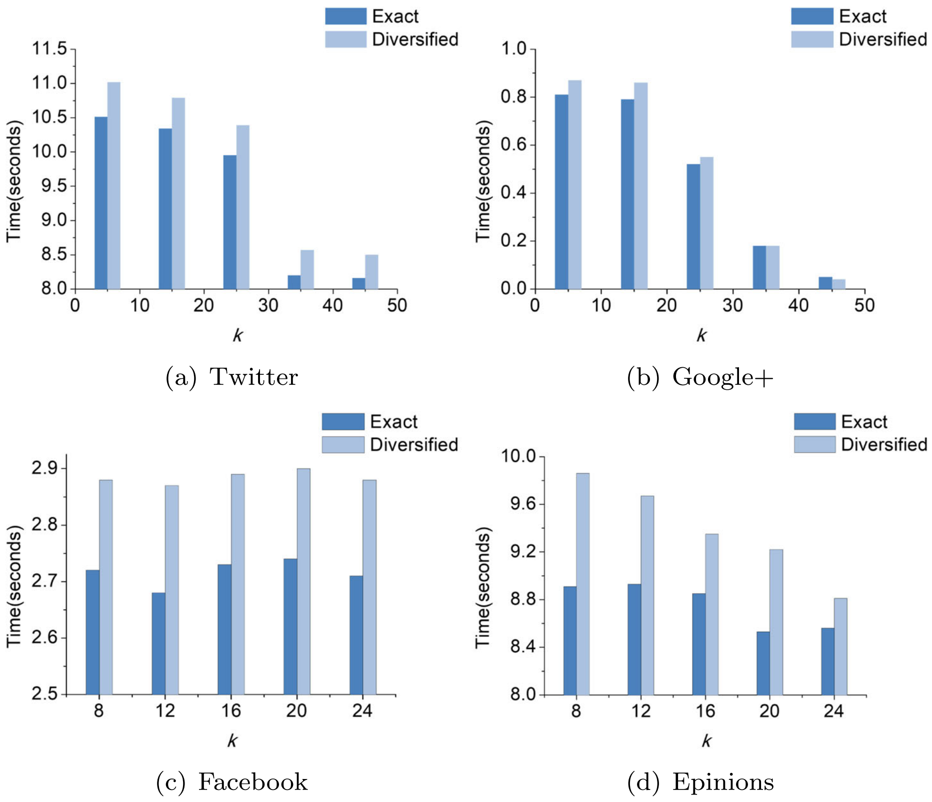
Figure 4 Similarity between Communities

C. *Evaluation of the maximal pk-Clique model*

The Label Propagation Algorithm (LPA) is considered to be an accurate model in detecting communities in social networks [32]. Wang et al. [32] also proposed Normalized Mutual Information (NMI) [9] as a popular criterion for evaluating the accuracy of community

Table 14 Similarity of top-8 influential communities

Top	Twitter		Google+		Facebook		Epinions	
1	–	–	–	–	–	–	–	–
2	1	0.83	1	0.90	1	0.86	1	0.42
3	1	0.91	2	0.97	1	0.86	2	0.91
4	1	0.91	3	0.95	1	0.86	1	0.91
5	2	0.91	2	0.97	3	0.88	1	0.73
6	3	0.91	5	0.97	5	0.95	5	0.91
7	1	0.52	6	0.94	5	0.95	5	0.91
8	7	0.91	5	0.97	6	0.95	6	0.91



(a) Twitter (b) Google+

(c) Facebook (d) Epinions

Figure 5 Comparision between exact and diversified search

detection models. NMI is based on defining a confusion matrix N, where the rows correspond to the “real” communities, and the columns correspond to the communities found by other method. The score of NMI stands for the agreement of two results. We identify communities using both LPA in the search space defined in Section 4.1 as the desirable communities. Then we compute the score of NMI for the results of our maximal *pk*-Clique model and the LPA model in this experiment. The scores we achieved were about 83% for Twttier, 58% for Google+, 67% for Facebook, and 63% for Epinions. The higher scores in the experiment demonstrate that our proposed community model is reliable in practice.

Table 15 Evaluation for different *l*

l	0.1	0.2	0.3	0.4	0.5	0.6
Twitter	10.38	10.45	10.45	10.40	10.36	10.42
Google+	0.84	0.84	0.84	0.83	0.83	0.83
Facebook	2.89	2.91	2.89	2.89	2.88	2.87
Epinions	9.39	9.32	9.30	9.42	9.32	9.29

Table 16 Aggregated influence of top-6 communities

Top	1	2	3	4	5	6
Exact	0.91	0.83	0.79	0.78	0.78	0.77
Diversified	0.91	0.83	0.79	0.78	0.76	0.74

D. Efficiency of proposed algorithms

In this and following subsections, we conduct experiments to evaluate the effectiveness of exact algorithms introduced in Section 6. We implement three versions of exact top-n search algorithm. The first one is pruning only, denoted as PS (Pruning based top-n ICS). The second is pruning and filtering, which is denoted as FS (Filtering plus pruning top-n ICS). The third is HS (Heuristic based top-n ICS including pruning and filtering). Tables 4, 5, 6 and 7 illustrate the results (time in seconds) obtained from different datasets, where BS denotes Basic Search algorithm.

It is clear that proposed pruning based PS, pruning plus filtering FS and heuristic based HS algorithms perform far more efficient than the basic solution, both in time and space costs. It also shows that the FS and HS perform better than PS in most cases. However, HS only outperforms FS in cases of searching for larger size cliques. We notice that the time cost does not decrease dramatically as vertices are progressively removed.

Tables 8, 9, 10 and 11 compare the space cost (in kbytes) of different algorithms. The results show that the PS, FS and the HS require less memory than the BS in each case.

E. Evaluation of scalability

To further evaluate the performance of proposed algorithms, we enlarg the search space over Facebook and Epinions datasets. The results are presented in Tables 12 and 13. The cost increases as the number of vertices increases in both cases.

F. The necessity to l-diversify

Figure 4 presents the redundancy over different maximal pk-Clique sizes. It indicates that the similarity increases as the size of maximal clique grows. The similarity is calculated according (14), where C_i and C_j represent any two maximal pk-Cliques.

In order to show the redundancy in the discovered top-n results at different values of n. We list top-8 maximal pk-Cliques found in above four datasets in Table 14, where k=20 for Twitter and Google+, k=8 for Facebook and Epinions. Take Twitter dataset as an example, in the second row, “1” means top-1 is the most similar to top-2 community, about 83%

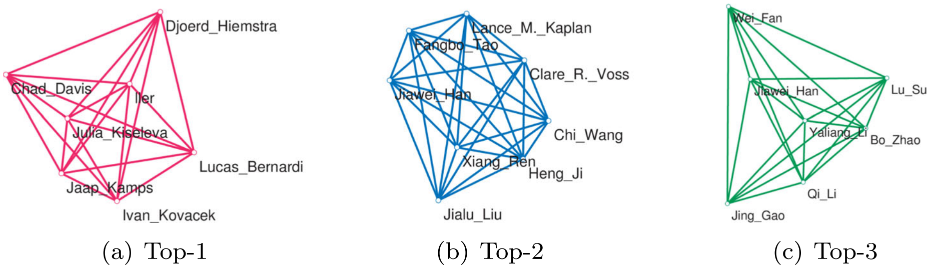


Figure 6 Top-3 influential communities for Alexanderm T

Table 17 Top 4-6 influential communities from diversified algorithm

Top 4	Top 5	Top 6
C_R._Voss	Jae-Gil_Lee	Alice_Leung
Fangbo_Tao	Jiawei_Han	Jiawei_Han
Heng_Ji	Lu_An_Tang	Lu_An_Tang
Jiawei_Han	Ming_Ji	Sangkyum_Kim
L_M._Kaplan	Roland_Kays	T_F._La_Porta
Taylor_Cassidy	Yintao_Yu	Wen-Chih_Peng
Xiang_Ren	Zhenhui_Li	Xiao_Yu

vertices are the same. In the eighth row, "7" means top-7 is most similar with top-8 cliques, they share about 91% common vertices. This shows the necessity to verify the query results.

$$Similarity_{C_{ij}} = \frac{\mid C_i \cap C_j \mid}{\mid C_i \cup C_j \mid} \tag{14}$$

G. Efficiency of diversified search

Figure 5 illustrates the results of exact and diversified top-n ICS. The diversify ratio is set to 0.4 in these experiments. It is observed that the cost of diversified search in those cases does not exceed 9% of exact algorithm. The cost is affordable in practice. We can take diversified top-n ICS as an optional option like heuristic search in practice.

Table 15 shows the results from different settings of l. There is no dramatically increase or decrease of time cost when vary the value of l.

H. Case Study

We construct a co-author network from the DBLP (DataBase systems and Logic Programming) data set for this case study. The diameter of the network is 6. A vertex represents an author and an edge is added between two authors if they are co-authors. The propagation probability is calculated according to the frequency of co-authorships. We perform a top-6 influential communities (ε=0.2, p=0.8, k=5) query for "Alexanderm T", the first author in the alphabet order. The threshold of diversity is set to 0.5.

The results are shown in Figure 6. It is fortunate to obtain the same top-3 from both exact and diversified algorithms. However, for top-4 to top-6, the results area different, as shown in Tables 17 and 18. We also present aggregated influence of top-6 communities from both

Table 18 Top 4-6 influential communities from exact algorithm

Top 4	Top 5	Top 6
C_R._Voss	Chi_Wang	Chi_Wang
Fangbo_Tao	C_R._Voss	Fangbo_Tao
Heng_Ji	A_El-Kishky	Heng_Ji
Jiawei_Han	Fangbo_Tao	Jialu_Liu
L_M._Kaplan	Heng_Ji	Jiawei_Han
Taylor_Cassidy	Jiawei_Han	Marina_Danilevsky
Xiang_Ren	Xiang_Ren	Nihit_Desai

algorithms. The results are shown in Table 16. It is easy to find out that communities top-4th, top-5th and top-6th from exact algorithm have similar influence (about 0.77-0.78), which indicate that they share more than 60% members.

8 Conclusions

In this paper, we study the personalized top-n influential community search problem in social networks. This kind of search provides a useful tool for other analytical tasks.

In particular, we propose a novel community model based on the maximal pk-Clique concept which uses a new cohesive criterion. To enable efficient personalized top-n influential community searches, we first present a pruning approach which prunes vertices according an upper bound of aggregated influence and a lower bound k from the pk-Cliques definition. Then we introduce search space refinement which removes unqualified vertices in the network before searching. We also propose a heuristic search approach. To diversify the results, we propose a diversified searching algorithm which returns results in one pass. A novel tree-like index and necessary operations are devised to accelerate the search process. Extensive experiments over real social networks verify the effectiveness and efficiency of our search algorithms.

Acknowledgments This work is supported by the National Natural Science Foundation of China (No. 61572165, 61803135), the Natural Science Foundation of Zhejiang Province (No. LZ15F 020003). Xiaoyi Fu's work is supported by Hong Kong Research Grants Council (No. 12200817, 12201615 and 12258116).

References

1. Akkoyunlu, E.A.: The enumeration of maximal cliques of large graphs. Siam J Comput **2**(1), 1–6 (1973)
2. Bron, C., Kerbosch, J.: Algorithm 457: finding all cliques of an undirected graph. Commun ACM **16**(9), 575–577 (1973)
3. Chang, L., Yu, X., Qin, L.: Fast maximal cliques enumeration in sparse graphs. Algorithmica **66**(1), 173–186 (2013)
4. Chen, L., Liu, C., Zhou, R., Li, J., Yang, X., Wang, B.: Maximum co-located community search in large scale social networks. Proc VLDB Endowm **11**, 1233–1246 (2018)
5. Chen, S., Fan, J., Li, G., Feng, J., Tan, K.l., Tang, J.: Online topic-aware influence maximization. Proc VLDB Endow **8**(6), 666–677 (2015)
6. Chen, W., Wang, C., Wang, Y.: Scalable influence maximization for prevalent viral marketing in large-scale social networks. In: Acm Sigkdd International conference on knowledge discovery and data mining (2010)
7. Cheng, J., Ke, Y., Fu, W.C., Yu, J.X., Zhu, L.: Finding maximal cliques in massive networks. Acm Trans Datab Syst **36**(4), 1–34 (2011)
8. Cui, W., Xiao, Y., Wang, H., Lu, Y., Wang, W.: Online search of overlapping communities. In: Proceedings of the ACM SIGMOD, pp. 277–288 (2013)
9. Danon, L., Duch, J., Díaz-Guilera, A., Arenas, A.: Comparing community structure identification (2005)
10. Eppstein, D., Maarten, L., Strash, D.: Listing all maximal cliques in sparse graphs in near-optimal time. Comput Sci **6506**, 403–414 (2010)
11. Fang, Y., Cheng, R., Luo, S., Hu, J.: Effective community search for large attributed graphs. Proc Vldb Endowm **9**(12), 1233–1244 (2016)
12. Fei, B., Chang, L., Lin, X., Zhang, W.: An optimal and progressive approach to online search of top-k important communities. Proc Vldb Endowm, 11(9) (2018)
13. Focacci, L.: MILANO: cost-based domain filtering. In: International conference on principles and practice of constraint programming (1999)
14. Goldenberg, J., Libai, B., Muller, E.: Talk of the network: a complex systems look at the underlying process of word-of-mouth. Mark Lett **12**(3), 211–223 (2001)

15. Granovetter, M.S.: Threshold models of collective behavior (1978)
16. Huang, X., Cheng, H., Qin, L., Tian, W., Yu, J.X.: Querying k-truss community in large and dynamic graphs. In: Proceedings of the ACM SIGMOD, pp. 1311–1322 (2014)
17. Huang, X., Lakshmanan, L.VS., Xu, J.: Community search over big graphs: models, algorithms, and opportunities. In: IEEE International conference on data engineering, pp. 1451–1454 (2017)
18. Huang, X., Lakshmanan, L.VS., Xu, J.: Community search over big graphs: models, algorithms, and opportunities. In: Proceedings of the ICDE, pp. 1451–1454 (2017)
19. Kempe, D., Kleinberg, J., Tardos, E.: Maximizing the spread of influence through a social network. In: Proceedings of the ACM SIGKDD, pp. 137–146 (2003)
20. Lee, J.R., Chung, C.W.: A query approach for influence maximization on specific users in social networks. IEEE Trans Knowl Data Eng **27**(2), 340–353 (2015)
21. Li, J., Liu, C., Yu, J.X., Chen, Y., Sellis, T., Culpepper, J.S.: Personalized influential topic search via social network summarization. In: Proceedings of the ICDE, pp. 17–18 (2017)
22. Li, J., Wang, X., Deng, K., Yang, X., Sellis, T., Yu, J.X.: Most influential community search over large social networks. In: Proceedings of the ICDE, pp. 871–882 (2017)
23. Li, R.H., Yu, J.X., Mao, R.: Efficient core maintenance in large dynamic graphs. IEEE Trans Knowl Data Eng **26**(10), 2453–2465 (2014)
24. Mcauley, J., Leskovec, J.: Learning to discover social circles in ego networks. In: International conference on neural information processing systems (2012)
25. Newman, M.E., Girvan, M.: Finding and evaluating community structure in networks. Phys Rev E **69**(2), 026113 (2004)
26. Ruan, J., Zhang, W.: An efficient spectral algorithm for network community discovery and its applications to biological and social networks. In: Seventh IEEE international conference on data mining, pp. 643–648 (2007)
27. Satuluri, V., Parthasarathy, S.: Scalable graph clustering using stochastic flows: applications to community discovery. In: Proceedings of the ACM SIGKDD, pp. 737–746 (2009)
28. Schmidt, M.C., Samatova, N.F., Thomas, K., Park, B.H.: A scalable, parallel algorithm for maximal clique enumeration. J Parallel Distrib Comput **69**(4), 417–428 (2009)
29. Shuang, Q., Jian, C., Xi, Z., Niu, B., Lu, H., Shuang, Q., Jian, C., Xi, Z., Niu, B., Lu, H.: Community discovering guided cold-start recommendation: a discriminative approach. In: IEEE International conference on multimedia and expo (2014)
30. Tomita, E., Tanaka, A., Takahashi, H.: The worst-case time complexity for generating all maximal cliques and computational experiments. Theor Comput Sci **363**(1), 28–42 (2006)
31. Wang, J., Cheng, J., Fu, W.C.: Redundancy-aware maximal cliques. In: Proceedings of the ACM SIGKDD, pp. 122–130 (2013)
32. Wang, M., Wang, C., Yu, J.X., Zhang, J.: Community detection in social networks: an in-depth benchmarking study with a procedure-oriented framework. Proceed Vldb Endow **8**(10), 998–1009 (2015)
33. Yuan, L., Qin, L., Lin, X., Chang, L., Zhang, W.: Diversified top-k clique search. In: 2015 IEEE 31st international conference on data engineering (ICDE), vol. 00, pp. 387–398 (2015)
34. Yue, W., Meliou, A., Miklau, G.: Rc-index: diversifying answers to range queries. Proc Vldb Endow **11**(7), 773–786 (2018)
35. Zhou, J., Fan, J., Wang, J., Wang, X., Li, L.: Cost-efficient viral marketing in online social networks. World Wide Web (2018)

Affiliations

Jian Xu[1] · **Xiaoyi Fu**[2] · **Yiming Wu**[1] · **Ming Luo**[1] · **Ming Xu**[1] · **Ning Zheng**[1]

Jian Xu
jian.xu@hdu.edu.cn

Yiming Wu
ymwu@hdu.edu.cn

Ming Luo
mingl@hdu.edu.cn

Ming Xu
mux@hdu.edu.cn

Ning Zheng
nzheng@hdu.edu.cn

[1] Hangzhou Dianzi University, Hangzhou, China

[2] Hong Kong Baptist University, Hongkong, China