2017 IEEE 33rd International Conference on Data Engineering

Most Influential Community Search over Large Social Networks

Jianxin Li
University of Western Australia
Perth, Australia
Email: jianxin.li@uwa.edu.au

Xinjue Wang
RMIT
Melbourne, Australia
Email: xinjue.wang@rmit.edu.au

Ke Deng
RMIT
Melbourne, Australia
Email: ke.deng@rmit.edu.au

Xiaochun Yang
Northeastern University
Shenyang, China
Email: yangxc@mail.neu.edu.cn

Timos Sellis
Swinburne University of Technology
Melbourne, Australia
Email: tsellis@swin.edu.au

Jeffrey Xu Yu
The Chinese University of Hong Kong
Hongkong, China
Email: yu@se.cuhk.edu.hk

***Abstract*—Detecting social communities in large social networks provides an effective way to analyze the social media users' behaviors and activities. It has drawn extensive attention from both academia and industry. One essential aspect of communities in social networks is outer influence which is the capability to spread internal information of communities to external users. Detecting the communities of high outer influence has particular interest in a wide range of applications, e.g., Ads trending analytics, social opinion mining and news propagation pattern discovery. However, the existing detection techniques largely ignore the outer influence of the communities. To fill the gap, this work investigates the *Most Influential Community Search* problem to disclose the communities with the highest outer influences. We firstly propose a new community model, *maximal kr-Clique community*, which has desirable properties, i.e., *society*, *cohesiveness*, *connectivity*, and *maximum*. Then, we design a novel tree-based index structure, denoted as C-Tree, to maintain the offline computed r-cliques. To efficiently search the most influential communities, we also develop four advanced index-based algorithms which improve the search performance of non-indexed solution by about 200 times. The efficiency and effectiveness of our solution have been extensively verified using six real datasets and a small case study.**

I. Introduction

Along with the rapid development of online social networks (e.g., Twitter, Facebook, Flickr), there is growing research interest to community detection which have been widely studied previously over biological networks and information networks (e.g., [12], [18], [22], [23]). The community detection discloses the structure of social networks and it supports analyzing the social media users' behaviors and activities. Conventionally, a community is defined as a densely connected subgraph with less connections to other nodes in the networks. However, it is insufficient in many applications where some particular aspects of communities are concerned. One essential aspect is the outer influence of community which is the capability to spread internal opinion/style/trend to external users of the community in social networks. In particular, the communities with high outer influence have opportunities to sway more external users into the new members of the community, the newly joined members will further increase the community's influence in the social networks. Various applications have interest in the outer influence of communities including Ads trending analytics, social opinion mining and news propagation pattern discovery. However, the outer influence of communities are largely ignored by existing detection techniques. To fill the gap, this work investigates the *Most Influential Community Search* problem to reveal the communities with the highest outer influences. The following are two examples where the most influential communities can be found.

Example 1: (Cold-start Community Recommendation) Cold-start problem is a significant problem in recommendation systems [21]. Assume that Miss Ellen is a new Twitter user. She would like to join in a music fan community to make online friends with common interest. While there are numerous such communities, it is difficult to recommend which one suits Miss Ellen because the preference of a new user is unknown to the recommendation system. In this situation, the established communities with high outer influence is ideal for Miss Ellen. It implies that the communities contains at least k members (i.e., society) and have main-stream opinion/trend/style in the social network (i.e., influential).

Example 2: (Trends Monitoring) Assume that a music distribution company wants to know the interest of Twitter users about the newly released songs. An effective way is to identify the music fan communities with the maximum outer influence in Twitter. The company can analyze the opinions of the community members to the new songs since their opinion are likely to become the main stream of all audiences; the company can also estimate how the external users are influenced by the influential communities to predict the trend. The information will help the company to review their marketing strategies and enable evidence driven decision-making.

The foundation of most influential community search is the capability to properly measure community outer influence. Typically, it is evaluated as the probability that the community influences the external users in social networks based on some influence propagation model, i.e., the *Linear Threshold* model (LT-model) [13] in this work. In addition to outer influence measurement, the most influential community is expected to possess proper inner properties. To this end, a novel explicit community model is defined based on the concept of maximal cliques [1], called *maximal kr-Clique community*. The maximal kr-Clique community model holds four desirable

2375-026X/17 $31.00 © 2017 IEEE
DOI 10.1109/ICDE.2017.136

inner properties, i.e., *society, cohesiveness, connectivity*, and *maximum*. Specifically, a maximal kr-Clique community is a connected and induced subgraph containing at least k nodes (*society*); the shortest path between any two nodes in a kr-Clique community are connected (*connectivity*) and the distance is not longer than r hops (*cohesiveness*); and a kr-Clique community is not a subgraph of another kr-Clique community (*maximum*). The maximal kr-clique community overcomes the drawbacks of the conventional community models, i.e., k-core [18] and k-truss [12]. For k-core, the community may be lack of cohesiveness because the hops of two nodes in a community may be very large. For k-truss, the connectivity requirement in a community is too strong to be practical in discovering reasonably large communities to form societies.

The most influential community search problem is very different from the *influence maximization* problem [4], [14], [16]. The influence maximization aims to select a certain number of nodes as seeds in social networks such that the number of other nodes activated by the seed nodes is maximized. It does not consider the structural constraints among the seed nodes, i.e., the seed nodes may be loosely connected and distant in the social networks. But in the most influential community, we are interested in the nodes which meet structural constraints, i.e., densely connected and within r hops in the social networks. In other words, the most influential community concerns the influence at the community level rather than the individual level.

Being an NP-hard problem, the most influential community search is highly challenging. We have deliberately designed a tree-based index, called *C-Tree*, to support efficient influential community search online. *C-Tree* is incrementally maintained when a community adds/deletes a user or a link. Also, we have developed four algorithms to efficiently find the most influential communities with the support of C-Tree. In particular, *C-Tree* and the proposed search algorithms are independent of community models. They can support the most influential community search where the other community models, e.g., k-core and k-truss, are applied. The contributions of this work are summarized as follows:

- We introduced the outer influence of community in social networks. It shades light to community detection when cogitating additional aspects of communities beyond structural constraints.
- We presented the innovative *maximal kr-clique community* which possesses desirable inner properties not held by conventional community model, i.e., k-core and k-truss.
- We designed a tree-based index (*C-Tree*) to effectively maintain the large number of communities in social networks.
- We have developed robust solutions to enable influential community search online. The efficiency and effectiveness of our solution have been extensively verified using six real datasets and a small case study.

The remainder of this paper is organized as follows. First, we propose and define the most influential community search problem in social networks in Section II and provide the baseline solution to the problem in Section III. Then, four index-based algorithms are presented to find the most influential maximal kr-Clique communities with the support of proposed *C-Tree* in Section IV. In Section V, we discuss the details of *C-Tree* index construction and update. The experimental results are analyzed in Section VI. The related work are concisely discussed in Section VII. Finally, this work is concluded in Section VIII.

II. Problem Definition

A social network is modeled as a directed graph $G = (V, E, P)$ where V is a set of nodes representing a set of social media users, E is a set of edges between nodes representing user-to-user relationships, and P is a set of weights associated with edges in E. The edge from node u to node v is denoted as (u, v) and the weight associated with edge (u, v) is denoted as P_{uv} to represent the probability that v is influenced by u.

Definition 1: (kr-Clique Community) Given a social network $G = (V, E, P)$, and two parameters k and r, a kr-clique community is an induced subgraph $C^{kr} = (V^{kr}, E^{kr})$ of G that meets the following constraints:

- Society - C^{kr} contains at least k nodes;
- Cohesiveness and Connectivity - any two nodes in C^{kr} can be reached at most r hops via the shortest path in the subgraph C^{kr}. Thus, it always holds that C^{kr} must be connected.

A kr-clique community C^{kr} is a *maximal* kr-clique community if C^{kr} is not a subgraph of any another kr-clique community $C^{kr'}$.

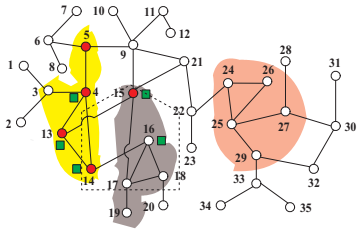


Fig. 1. An example social network graph.

Example 3: Consider a small social network as shown in Figure 1. Suppose $r = 2$ and $k = 4$. The subgraph induced by nodes $\{3, 4, 5, 13, 14\}$ is a maximal kr-clique community based on Definition 1. Although the subgraph induced by nodes $\{3, 4, 5, 13\}$ also satisfies the constraints related to k and r, it is not a maximal kr-clique community because it is contained in another kr-clique community induced by nodes $\{3, 4, 5, 13, 14\}$. There are 19 maximal kr-clique communities among which six contain five nodes. The six communities have been annotated by different colors or line styles as shown in Figure 1.

In this paper, the widely-accepted influence propagation model, i.e., Linear Threshold (LT) [4], [13], is adopted. To measure the outer influence of a community C, only the member nodes in C are activated initially. At the end of influence propagation process, any external node v is successfully activated if and only if the *aggregated probability* that

v is influenced by the member nodes in C is not less than a threshold Δ.

Definition 2: (Aggregated Influence Probability)

$$Pr(v|V(C)) = 1 - \prod_{u \in V(C)} (1 - P_{u \to v}). \quad (1)$$

where $V(C)$ is the set of member nodes in community C and $P_{u \to v}$ is the probability that v is influenced by u. It is computed by multiplying the influence probabilities of the edges in the maximum influence path from u to v. The maximum influence path has been verified to be an effective way for evaluating the influence between users in [15].

Definition 3: (Outer Influence of a Community) Given a social network $G = (V, E, P)$ and a community C where only the member nodes in C are activated initially, the outer influence of C is the number of external nodes to be successfully activated by C at the end of influence propagation process in G. Formally:

$$Score(C) = |\{v | v \in V(G) \backslash V(C) \wedge Pr(v|V(C)) \geq \Delta\}| \quad (2)$$

The activation threshold Δ is a system parameter which can be learned from historical responses of social media users [11].

Example 4: Consider a maximal kr-clique community induced by nodes {3, 4, 5, 13, 14} in Figure 1. To make the example concise, we assume that each edge is associated with equal influence probability (i.e., 0.5), and we ignore the pairs of nodes where the probability that one node is influenced by the other node is less than 0.15. We set Δ at 0.4. So, we get the following influenced node list for each member node in the community.

Node	Influenced Nodes with Probability
3	(1, 0.5), (2, 0.5)
4	(1, 0.25), (2, 0.25), (6, 0.25), (9, 0.25), (15, 0.25), (16, 0.25)
5	(6, 0.5), (7, 0.25), (8, 0.25), (9, 0.5), (10, 0.25), (11, 0.25), (15, 0.25), (21, 0.25)
13	(9, 0.25), (15, 0.5), (16, 0.25), (17, 0.25), (21, 0.25)
14	(15, 0.25), (16, 0.5), (17, 0.25), (18, 0.5)

Nodes {1, 2, 6, 9, 15, 16} will be activated by some individual member node of community {3, 4, 5, 13, 14}. Nodes {17, 21} will be activated by the aggregated influence from multiple member nodes of community {3, 4, 5, 13, 14}. Node 17 (or 21) is influenced with probability $1-(1-0.25)(1-0.25) = 0.44$ by node 13 and 17 (or node 5 and 13). Therefore, the outer influence score of the community is 8.

Therefore, the target of this work is to search for the maximal kr-cliques that have the maximum outer influence scores in social networks. We refer the problem as *most influential community search* in this paper.

Definition 4: (Most Influential Community Search) Given a social network $G = (V, E, P)$ and an integer k, our target is to find the maximal kr-clique community C satisfying:

$$\arg\max_{V(C) \subseteq V(G)} Score(C) \quad (3)$$

subject to

$$|V(C)| \geq k, \forall u, v \in V(C) \quad |sp(u, v)| \leq r$$

where $V(C)$ is the set of vertices in community C (i.e., $|V(C)|$ is the set size), $sp(u, v)$ represents the shortest path between the vertices u and v (i.e., $|sp(u, v)|$ is the hops of the shortest path $sp(u, v)$), and r is a system parameter.

To determine the value of r, the statistics of communities in social networks have been studied and presented in Figure 3. When $r = 1$, most communities only contain less than 10 nodes and when $r = 3$, 95% of the nodes in social networks present in at least two communities. Since we are interested in the relative large communities with less overlap, it only makes sense for r to take the value of 2. This parameter will be further discussed in Section VI.

There may be more than one most influential communities to be returned when several maximal kr-cliques have the same maximum outer influence score. As discussed in [9], the maximal clique problem (MCP) has been proved to be an NP-hard problem. Our most influential community search problem is more complicated because we not only need to identify the maximal cliques with their size no less than k, but also compute their outer influences. As such, the proposed research problem is also an NP-hard problem.

III. Baseline Solution

To address the most influential community search problem, the basic solution is to first sort the nodes in $V(G)$ by their individual influence score and then check the sorted nodes one by one in the descending order. For each node, we need to compute its maximal kr-cliques and calculate the outer influence score of each found maximal kr-cliques. After all the nodes have been checked, we compare their outer influence scores and return the maximal kr-clique with the maximum score as the most influential community.

Algorithm 1 Basic Solution

Input: A social graph $G = (V, E, P)$ and an integer k
Output: A set $\mathbb{C}$ of Most influential maximal kr-clique communities

```
1: Offline compute the influence score of each node in V;
2: Get a list L with the nodes sorted by their influence scores;
3: Transform G to G_r using the system setting r;
4: for all u ∈ L do
5:     Compute the maximal kr-cliques containing u from G_r;
6:     Calculate the outer influence score of each maximal kr-clique;
7:     Update ℂ by writing the maximal kr-cliques with the maximum score;
8:     next_k_score ← add the next k nodes' individual influence score in L;
9:     if the score in ℂ > next_k_score then
10:        return ℂ;
11: return ℂ;
```

The baseline algorithm is presented in Algorithm 1. Here, the influenced node list and their influence score for each node have been offline computed and maintained. As such, the time complexity of the algorithm mainly consists of the following parts. The first part is to list all maximal cliques that would be $O(3^{n/3})$ using the state-of-the-art algorithm in [24] where n is the total number of nodes in the social network under consideration. Based on the community size distribution as shown in Figure 3, the number of generated k-communities would be in the complexity of $O(log_k n)$.

To evaluate the outer influence score of each community, we need to find the influence path from every community member node to its external nodes as discussed in [15]. The computational complexity would be $O(kn)$ where k is the community size and n is the social network size. Therefore, the time complexity of Algorithm 1 is $O(3^{n/3} + knlog_k n)$. From the complexity analysis, we can see that the bottlenecks of the algorithm are the maximal clique generation part and the outer influence computation part. Therefore, we focus on lifting the processing efficiency of these two parts by designing tailored index and advanced search algorithms.

IV. Index-based Influential Community Search

In this section, we first propose a tree structure, called C-tree, to index maximal r-cliques generated from G. And we show that *C-Tree* is compact and any kr-cliques can be generated efficiently by using *C-tree*. Then we propose four efficient search algorithms based on C-tree, which are *sequential-order based* (SO) search, *improved sequential-order based* (SO+) search, *best-first based* (BF) search, and *fast best-first based* (BF+) search. The first two algorithms evaluate the candidate communities supported by *C-Tree* in a sequential order, by which the influence computation of common nodes can be avoided. The last two algorithms probe the candidate communities based on the upper bounds of their outer influence. As a result, many nodes can be filtered out in the influence computation.

A. Indexing Maximal r-Cliques

According to the discussion in Section III, the baseline solution is too expensive to be practical because there is a large number of maximal r-clique communities that need to be computed in social networks. In addition, the large number of communities often contain large overlaps across different communities. The repeated computation on such overlaps will incur significant overhead. Therefore, it is desirable for us to propose an effective tree index, called *C-Tree*, to precompute and maintain the large number of communities with the minimal community overlaps.

Definition 5: (*C-Tree*) Given a social network G and a system parameter r, we have all maximal r-cliques $\mathcal{C} = \{C_1, C_2, ..., C_{n_c}\}$ that may contain many duplicate nodes. The *C-Tree* $T(\mathcal{C})$ of $\mathcal{C}$ is defined as:

- The root of $T(\mathcal{C})$ is a virtual node;
- An internal node of $T(\mathcal{C})$ is the maximum intersection of all maximal r-cliques in the subtree rooted at the internal node. For any leaf node of $T(\mathcal{C})$, it does not have intersection with its siblings;
- Each path from the root of $T(\mathcal{C})$ to a leaf node is a maximal r-cliques community.

Figure 2 presents a *C-Tree* for the small social network in Figure 1. For example, the community induced by {3, 4, 5, 13, 14} can be identified from the most left path in Figure 2.

B. Sequential-Order based Search

Since each path from the root to the leaf node represents a community, we can directly access each path and retrieve

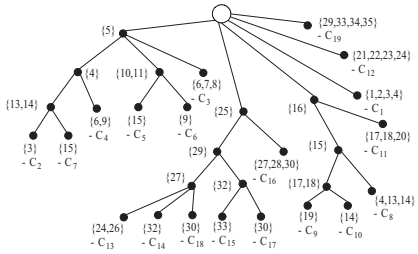


Fig. 2. Example of C-Tree

all the nodes in the corresponding community of the path. Obviously, some nodes will be re-accessed many times because they are shared by multiple communities. To avoid this, we propose a stack-based algorithm to access the nodes in the *C-Tree*, which maintains the shared nodes in a stack until all the communities containing them have been dealt with. As such, we can retrieve each community from the index tree in the depth-first-order and calculate its influence score based on Definition 3. We call the community search method as *Sequential Order based* search, denoted as *SO*.

Algorithm 2 SO Algorithm

Input: *C-Tree* T of a social network G and an integer k
Output: A set $\mathbb{C}$ of Most influential maximal kr-clique communities

```
1:  New two stacks A and B, set Score(C_max) as zero;
2:  Get the root node v_root of the C-Tree T;
3:  A.push(v_root);
4:  while A is not empty or B is not empty do
5:      u = A.pop();
6:      if B is not empty then
7:          if u is a child of B.top() and u.children()≠null then
8:              B.push(u);
9:              for v ∈ u.children() do
10:                 A.push(v);
11:         else if u is a child of B.top() and u.children()==null then
12:             if B.size≥ k-1 then
13:                 C_temp = B ∪ {u};
14:                 Score(C_temp) = Computing_Score(C_temp);
15:                 ℂ ← C_temp if Score(C_max) < Score(C_temp);
16:             else
17:                 B.pop();
18:     else
19:         B.push(u);
20:         for v ∈ u.children() do
21:             A.push(v);
22: return ℂ;
```

The detailed procedure is presented in Algorithm 2. Here, we utilize two stacks to implement the depth-first tree traversal. The tree traversal starts from the root node (i.e., v_{root}) of the index tree T. At Line 5, we get the node u at the top position of Stack A. And then we check if u can be added into Stack B, as shown in Line 7-Line 17. If u is a child of the node B.top() (i.e., the node at the top position in B) and it is also an internal node of T, i.e., u.children() $\neq$ null, then we just explore the children nodes of u in the tree T, as shown in Line 8-Line 10. Here, the node set in Stack B is only a subset of a community or multiple communities. If u has been identified as a leaf node, i.e., u.children()=null, then we can see that the nodes in B and node u are on the same path, i.e., it is a complete community C_{temp} based

on Definition 5. At this moment, we also need to check the community size. If the size of the community C_{temp} is not less than the required size k, we compute its influence Score(C_{temp}) by calling function Computing_Score(C_{temp}) and record the more influential community $\mathbb{C}$ via comparison, which is described in Line 12-Line 15. When u is not a child of the node $B.top()$, it says that all communities containing the node $B.top()$ have been processed. In this case, we only need to pop it out from Stack B. The algorithm will be terminated until all communities in the index tree have been visited. At the end, the most influential community $\mathbb{C}$ will be returned. Assume the *C-Tree* is an x-ary tree index and has h levels. The complexity of *SO* is $O(\frac{x^{h+1}-1}{x-1}kn)$ where $\frac{x^{h+1}-1}{x-1}$ is the total number of nodes in the tree and $O(kn)$ is the complexity of estimating a community's influence.

C. Improved Sequential-Order based Search

In this section, we propose two upper bound based pruning properties, which can be used to improve the efficiency of sequential-order based search by pruning more insignificant candidate communities, which is denoted as *SO+*.

Before providing the first upper bound, let's review the important property in influence maximization problem. This is because the influence of a community in this work is calculated by the maximum node set influenced by the community minus the community member set. The following property has been proved in many works [5], [6], [13].

Property 1: The influence maximization $f()$ is a monotonic and submodular function. I.e., for any two sets S and X, $f(S \cup X) + f(S \cap X) \leq f(S) + f(X)$ always holds.

Now, we can prove Property 2. Consider two communities C_1 and C_2. Their maximum common part is denoted as $C_{1,2} = C_1 \cap C_2$. To be simple, we let $S_1 = C_1 \setminus C_{1,2}$, $S_2 = C_2 \setminus C_{1,2}$, and $X = C_{1,2}$. Our target is to look for the condition that guarantees $f(S_1 \cup X)$ - $(S_1 \cup X)$ is equal to or larger than $f(S_2 \cup X)$ - $(S_2 \cup X)$ with less computation.

Property 2: (Tight Upper Bound) If $f(S_1|X)$ - $S_1 \geq f(S_2)$-S_2, then we can directly conclude that $f(S_1 \cup X)$ - $(S_1 \cup X) \geq f(S_2 \cup X)$ - $(S_2 \cup X)$ without computing $f(S_2 \cup X)$. it says that C_1 is more influential than C_2. Therefore, compared with C_2, C_1 is only required to be maintained as the most influential candidate communities.

Proof: Since X is not a subset of S_1, there is no overlap between S_1 and X. So $S_1 \cup X$ can be rewritten as $S_1 + X$. $f(S_1 \cup X)$ - $(S_1 \cup X)$ - $[f(S_2 \cup X)$ - $(S_2 \cup X)]$ can be rewritten as $f(S_1 \cup X)$ - $(S_1 + X)$ - $[f(S_2 \cup X)$ - $(S_2 + X)]$ = $f(S_1 \cup X)$ - S_1 - $f(S_2 \cup X) + S_2$. Based on Property 1, we have $f(S_2 \cup X) \leq f(S_2) + f(X)$ - $f(S_2 \cap X) = f(S_2) + f(X)$ where $f(S_2 \cap X)$ is zero because $S_2 \cap X$ is null. Therefore, we have $f(S_1 \cup X)$ - S_1 - $f(S_2 \cup X) + S_2 \geq f(S_1 \cup X)$ - S_1 - $(f(S_2) + f(X)) + S_2 = f(S_1 \cup X)$ - $f(X)$ - S_1 - $f(S_2) + S_2 = f(S_1|X)$ - S_1 - $(f(S_2)$-$S_2)$, which satisfies our pre-condition. ■

Property 3: (Loose Upper Bound) Consider any set of nodes, e.g., $S = \{u\}$ where S is the set of member nodes in community C_i. $R(u)$ represents the reachable node set of node u. The upper bound of the community's influence $(f(S) - S)$ for S is $|\{\cup R(u) | u \in C_i\} \setminus C_i|$, denoted as UB($S$) or UB($C_i$).

According to Property 2 and Property 3, we can filter out the insignificant candidate communities via partial evaluation.

Algorithm 3 SO+ Algorithm

```
1: The same codes of Line 1-Line 11 in Algorithm 2;
2: if B.size≥ k-1 then
3:     C_temp = B ∪ {u};
4:     if Score(ℂ) ≥ UB(C_temp) then
5:         Do nothing;
6:     else
7:         for all C_max ∈ ℂ do
8:             X = C_max ∩ C_temp;
9:             if Score(C_max) ≥ Computing_Score(C_temp \ X) then
10:                Do nothing;
11:            else
12:                Score(C_temp) = Computing_Score(C_temp);
13:                ℂ: C_max ← C_temp if Score(C_max) < Score(C_temp);
14: The same codes of Line 16-Line 21 in Algorithm 2;
15: return ℂ;
```

The detailed procedure of the improved sequential-order based search is presented in Algorithm 3. Different from Algorithm 2, Algorithm 3 will check the pruning conditions, i.e., the loose upper bound and the tight upper bound. If the new generated community can be filtered out by using the upper bounds, then we do not need to compute its influence score. The efficiency can be improved due to the following two main reasons: (1) if we know some nodes can be activated above a threshold by $C_{temp} \setminus X$, these nodes must be successfully activated above the same threshold by its superset C_{temp}. Therefore, we can exclude these nodes from the computing process of Computing_Score(C_{temp}); (2) Even if we cannot filter out the community C_{temp} by the partial result of Computing_Score($C_{temp} \setminus X$), the intermediate results of Computing_Score($C_{temp} \setminus X$) can continuously be used to make calculation for Computing_Score(C_{temp}). We don't need to do Computing_Score(C_{temp}) from scratch. The complexity of *SO+* is $O(\frac{x^{h+1}-1}{x-1}kn)$. But the upper bound based pruning rules help to filter out the insignificant communities so as to improve the efficiency.

D. Best-First based Search

The performance of SO+ is sensitive to the influence score of the first probed community because if the first community is lucky to be the most influential one, then it can improve the efficiency a lot. Otherwise, it has less pruning power, e.g., no pruning can be applied when the least influential one appears at the first place. To effectively access *C-Tree* with a "good" order, we develop a robust best-first search algorithm (denoted as *BF*) in this section. From Property 3, we know that for node u in a community C_i and the reachable node set $R(u)$ of u, the influence of C_i is bounded by $|\{\cup R(u) | u \in C_i\} \setminus C_i|$, denoted as UB($C_i$).

Lemma 1: Assume we have computed the influence $Score(C_1)$ of a community C_1 based on Definition 3. For any other communities C_i that has not been visited, we can skip C_i without computation if $Score(C_1) \geq$ UB(C_i).

The key idea of BF approach is to find all candidate communities with no community size less than k from the *C-Tree*. And then it sorts these communities by their upper bound values and maintain them in an ordered queue Q. Every time,

we pop the candidate community C_x from the top position of Q, which should have the highest upper bound value. We then calculate its influence score. If the computed score of C_x is not less than the upper bound of the next community in Q, then we keep C_x as the currently most influential community. Otherwise, we need to use the calculated real score to replace the upper bound value of C_x, and add it back to Q. The process will be repeated until the most influential community is found.

Algorithm 4 BF Algorithm

Input: *C-Tree* T of a social network G and an integer k
Output: A set $\mathbb{C}$ of Most influential maximal kr-clique communities

```
1: Initialize two stacks A and B, and a queue Q;
2: The same codes of Line 2-Line 11 in Algorithm 2;
3: if B.size≥ k-1 then
4:    C_temp = B ∪ {u};
5:    C_temp.ub = UB(C_temp);
6:    C_temp.score = null;
7:    Add C_temp into Q and sort the communities by upper bounds;
8: The same codes of Line 16-Line 21 in Algorithm 2;
9: found = false, ℂ = null;
10: repeat
11:    C_x = Q.pop();
12:    if C_x.score is not null then
13:       ℂ ← C_x;
14:       found = true;
15:    else
16:       C_x.score = Computing_Score(C_x);
17:       if C_x.score ≥ Q.top().ub then
18:          ℂ_x;
19:          found = true;
20:       else
21:          C_x.ub = C_x.score;
22:          Add C_x back to the ordered queue Q;
23: until found
24: return ℂ;
```

The detailed procedure of best-first based search is presented in Algorithm 4. In Line 1-Line 8, we access the *C-Tree* and generate an ordered queue containing all the candidate communities with no size less than the parameter k. In Line 9-Line 23, we check the candidate communities based on their upper bound values in the queue. For a probed candidate community C_x, we compute its influence score at Line 16. Line 21 is used to update the upper bound value of a community using its real influence score that has been computed. The complexity of *BF* is $O(knlog(\frac{x^{h+1}-1}{x-1}))$ where $log(\frac{x^{h+1}-1}{x-1})$ is the complexity of probing the tree nodes via binary search.

E. Fast Best-First based Search

To further accelerate the efficiency of community search, in this section we develop the faster best-first based approach, which is denoted as *BF+*.

The key idea of *BF+* is to maintain a pre-computed list. The list contains the *C-Tree* leaf node IDs where each leaf node ID represents the corresponding community of the path from root to the leaf node. And the list is sorted by the upper bound value of the influence of the community. We compute and sort the list through offline computation. When a user issues a community search query by specifying the parameter k, we first access the list in the ascending order. For each leaf node, we can locate its corresponding community by traversing the path from the tree root to the leaf node in the C-Tree. Thus, the first visited community should have the highest upper bound value on the influence. And then, we calculate the real influence score of the community if the visited community contains no less than k members. Otherwise, we discard the community and probe the next community in the ordered list. After that, we compare the real influence score of the community with the upper bound value of the next community. If the real influence score of the community is not less than the upper bound value of the next one, then the current community is the most influential community at this moment. Otherwise, we use the real score of the community to replace its upper bound value. The correctness has been proved in Lemma 1. We repeat the above procedure until the termination condition is met. The algorithm is similar to the procedures in Line 9-Line 23 in Algorithm 4. The complexity of *BF+* is $O(knlog(x^h))$ where $O(log(\frac{x^{h+1}-1}{x-1}))$ is smaller than $O(log(x^h))$ because of $\frac{x^{h+1}-1}{x-1}$ - x^h = $\frac{x^h-1}{x-1} > 0$. This is because most of the time *BF+* only needs to access the number (i.e., x^h) of leaf nodes in the *C-Tree*.

V. *C-Tree* Index Construction

Now we show how to efficiently construct *C-Tree* index. We first show the procedure of enumerating all the maximal cliques for a social graph by revisiting the work in [24]. And then, we present the algorithm of index construction and discuss the complexity of building and maintaining the *C-Tree*.

A. Revisiting Maximal Cliques Enumeration

To generate the maximal r-cliques from a graph G, we first transform the graph into r-hop based graph G_r where we create a direct edge for any two nodes if their shortest distance is not more than r hops in G. Then, we recursively call a function Clique_Generation to produce the maximal r-cliques. We implement Clique_Generation by revisiting the state-of-the-art algorithm for maximal clique enumeration proposed in [24]. Its efficiency has been further verified in [8].

Algorithm 5 EnumkrMC()

Input: A social graph $G = (V, E)$ and a system parameter r
Output: maximal r-clique set $\mathcal{C}$

```
1: Transform G to G_r w.r.t. r;
2: C ← Clique_Generation(φ, V, φ);
3: Function Clique_Generation(node set R, node set P, node set X)
4: if P == φ and X == φ then
5:    if |R| ≥ 3 then
6:       add R as a maximal clique into C;
7: else
8:    u ← argmax_{v∈P∪X}{|P ∩ N_r(v)|};
9:    for all v ∈ P \ N_r(u) do
10:      num = num_ini;
11:      Clique_Generation(R ∪ {v}, P ∩ N_r(v), X ∩ N_r(v));
12:      P ← P \ {v};
13:      X ← X ∪ {v};
14: return C;
```

The detailed procedure is presented in Algorithm 5. In function Clique_Generation, it maintains three node sets: R, P and X. The nodes in R form a partial clique to be expanded. P contains the nodes that are adjacent to all the nodes in R

in the new graph G_r and are potential candidates to be added to the maximal clique. X contains the nodes such that (1) they are adjacent to all the nodes in R in the new graph G_r; and (2) they have been traversed to form maximal cliques in any previous level of recursion. Adding any node in X to the current partial clique R will result in duplicate cliques. Therefore, nodes in X must be excluded from R where P and X are used to cover the nodes that are adjacent to all nodes in R in the new graph G_r. When both P and X become empty (Line 4), R cannot be further expanded. Thus, we output R as a maximal clique (Line 5-Line 6), in which, we are only interested in the communities with more than two nodes. If R can be further expanded, then we recursively add a node from the candidate nodes in P to expand the current partial clique R. Each time a vertex v is added into R, we refine P and X by keeping only the nodes that are also adjacent to v (Line 11) and invoke the function Clique_Generation recursively. P and X are updated after each recursive call (Line 12-Line 13). The pivot node u in Line 8 is used to reduce the computational cost of Clique_Generation. In principal, every node in P or X can be chosen as a pivot node. Here, we choose the one with the maximum number of neighbors in P because such a pivot node is shown to be computationally effective in [8]. The complexity of Algorithm 5 is $O(3^{|V|/3})$ as discussed in [8].

B. Algorithm of Constructing C-Tree

Theorem 1: Constructing *C-Tree* is an NP-Hard problem.

Proof: Constructing *C-Tree* can be equivalently transformed to a minimization problem, i.e., minimizing $\sum_{n_1,n_2 \in T} |n_1 \cap n_2|$ where $\cup_{n_i \in T}\{n_i\} = V(G)$ and $\cup_{v_j \in path(n_i)}$ is a community. Therefore, the problem of constructing *C-Tree* can be reduced to the generalized assignment problem that is NP-hard. ■

By reviewing the procedure of generating maximal cliques in [24], we can see that the maximal cliques are generated in the recursive process and the recently generated cliques may have high chance to be overlapped. As such, the overlapped part is the frequent node set in the local social network portion covering the nearly maximal cliques. However, it cannot provide direct help to us because the sequence of generated cliques are very sensitive to the selection of the nodes to be probed in the procedure of clique generation [24]. We cannot assume it can probe the best node every time.

To address this challenge, we propose a novel technique to identify the local-frequent nodes, which is used to construct the *C-Tree* for a given set of communities.

Definition 6: (Node Duplicate Frequency NDF) Consider all maximal r-cliques $\mathcal{C} = \{C_1, C_2, ..., C_{n_c}\}$ in a social network G. For any node $u \in C_i$, its node duplicate frequency $NDF(u)$ is the number of communities in $\mathcal{C}$ containing node u.

Property 4: (Local Most Duplicate Node) Given a subset of communities $\mathcal{C}_{sub} \subseteq \mathcal{C}$, the local most duplicate node u in $\mathcal{C}_{sub}$ is the node with the maximum duplicate frequency in $\mathcal{C}_{sub}$, denoted as $\text{NDF}_{\mathcal{C}_{sub}}(u)$.

Assume we maintain the community information by using node lists, i.e., community ID C_i: member nodes $u_1, u_2, ...$ For each community, we can sort the nodes in the node list in descending order based on their NDF scores. After that, we can easily identify the most duplicate nodes in the node list. In the other words, if a node is highly duplicated in communities, then the node will appear at the top position in their community node lists. By doing this, we can easily select the most duplicate node and add it as an internal node into the *C-Tree*. We then exclude the node from the related communities and deal with these communities with the remaining nodes under the subtree rooted at the added internal node in the *C-Tree*. For the other communities that do not contain the node at their top positions, we will deal with them based on their shared nodes at their top positions in a similar way. At the next iteration, we re-calculate the local most duplicate node for each branch of communities based on Property 4. The *C-Tree* can be constructed level by level, while the duplicates can be maximally reduced.

Example 5: (*C-Tree*) Figure 2 presents the *C-Tree* for the small social network in Figure 1 based on Property 4. Here, we only maintain the communities with no less than four members. Firstly, since node 25 in Figure 2 has the most duplicate frequency, we create a new internal node $\{25\}$ including the communities $\{C_{13}, C_{14}, C_{18}, C_{15}, C_{17}, C_{16}\}$. And then, we deal with node 5 in Figure 2 due to its high duplicate frequency, which results in a new addition of node $\{5\}$. The new internal node includes the communities $\{C_2, C_7, C_4, C_5, C_6, C_3\}$. Similarly, we can calculate the duplicate frequencies of the nodes and select the new internal nodes for the remaining communities. After that, the first level of the tree is constructed. At the following iterations, we repeat the above procedures for each subset of communities and build the tree level by level. The above procedure will be terminated until there is no duplicate among sibling nodes in the tree.

Algorithm 6 CTree()

Input: Maximal r-clique community set $\mathcal{C} = \{C_1, C_2, ..., C_{n_c}\}$ and a tree root v_{root}
Output: a *C-Tree* T

1: **while** $\mathcal{C}$ is not empty **do**
2: Calculate the NDF score for each node in $\mathcal{C}$;
3: Sort the nodes in each community by their NDF scores;
4: Get node u where $NDF_{\mathcal{C}}(u) \geq NDF_{\mathcal{C}}(u'|u' \in \backslash u)$;
5: Add u as a child node into v_{root} in T;
6: Get the subset $\mathcal{C}_u$ of communities with u at their top positions and remove u from $\mathcal{C}_u$;
{Incur a recursive process for each subset of communities}
7: u.CTree($\mathcal{C}_u$, u);
8: $\mathcal{C} \leftarrow \mathcal{C} \setminus \mathcal{C}_u$;
9: **return** The tree T with the root node v_{root};

The detailed procedure is presented in Algorithm 6. The key idea of this algorithm is to select the most duplicate node u for splitting the communities. Obviously, the communities can be splitted into two sets: one set of communities contain u at their top positions, and another set of communities do not. For the first set, we take u and add it as a child node of the current root, as shown in Line 5. And we remove u from the community subset $\mathcal{C}_u$. After that, the similar procedure as above is used to deal with the new community subset $\mathcal{C}_u$ where u is taken as the new root. For the second set, i.e., $\mathcal{C} \setminus \mathcal{C}_u$, we repeat the steps in Line 2-Line 8. When all subsets become empty, the whole recursive algorithm can be terminated. At the

end, the *C-Tree* can be obtained via the root node v_{root}. The complexity of Algorithm 6 is $O(log_3 2n)$ where we assume all the communities with more than two nodes will be maintained and they are compared pairwise.

C. C-Tree Optimization and Update

To further reduce the duplicate nodes in C-Tree, we develop an iterative procedure to optimize the C-Tree. The key idea is to check the nodes with high duplicate frequency in the *C-Tree* built in Section V-B and identify each node that the optimal benefit of changing it is larger than the cost. For each node u to be checked, its optimal benefit can only be calculated based on the branches containing node u.

Example 6: (*C-Tree* Optimization) Take node 4 in Figure 2 as an example. We can see the related branches are C_2, C_7, C_4, C_8 and C_1 in Figure 2. If we group these branches based on node 4, then the optimal benefit is that we can reduce two duplicates of node 4 (coming from C_8 and C_1) and reduce one duplicate for nodes 13, 14, 15. Therefore, the total optimal benefit of changing node 4 is weighted as 5. Note that the optimal benefit is only the upper bound value of changing the node. Similarly, we also need to calculate the cost based on the branches containing node u. Consider node 4 in Figure 2 as an example again. If we group these branches based on node 4, then the cost is that we need to bring new duplicates for nodes 5, 15 and 16 because they are the ancestors of the tree nodes containing node 4. Therefore, the cost of changing node 4 is weighted as 3. Since the optimal benefit is larger than the cost, we will try to change the *C-Tree* by adjusting the position of node 4. If the exact benefit is still larger than the exact cost, then we update and generate a new version of *C-Tree*. Otherwise, we keep the *C-Tree* unchanged.

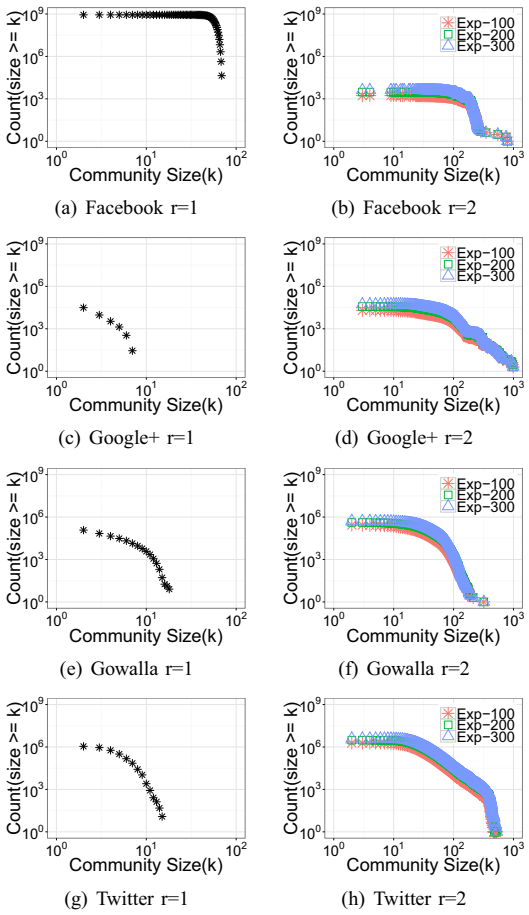
Algorithm 7 CTreeOptimization()

Input: A *C-Tree* T
Output: An optimized *C-Tree* T'

1: $S \leftarrow$ Get the nodes with duplicates in T;
2: **while** S is not empty **do**
3: $u = argmax_{v \in S}\{Benefit(v,T) - Cost(v,T)\}$;
4: **if** $Benefit(u,T) - Cost(u,T) > 0$ **then**
5: $T' \leftarrow Refine(T,u)$;
6: $S' \leftarrow$ Get the nodes with duplicates in T';
7: $S = S' \cap S$;
8: $S = S \setminus \{u\}$;
9: **return** The refined tree T';

Algorithm 7 presents the detailed procedure of optimizing *C-Tree*. Here, we use a candidate set S to maintain the nodes that need to be refined. At the beginning, S only contains the nodes with duplicates in the generated *C-Tree* T. We will reduce the candidate set S through iterations. At each iteration, we select node u that can bring the maximal gain to the tree refinement (Line 3). We refine the tree T into T' based on the adjustment of node u (Line 5). At the next iteration, we only need to consider the nodes that appear as the duplicate nodes in T' and T (Line 6-Line 7). The algorithm can be terminated until the candidate set becomes empty. Since we only check one time for each node at most, the complexity of this algorithm is $O(n)$.

The *C-Tree* can be updated incrementally when new users or links are added into the social network G. To do this, we first identify the maximal r-cliques for the new additions via nearby search in G. And then we find the node in the *C-Tree* where the tree node (i.e., a community) has the maximum overlap with the new maximal r-cliques. Finally, we add them into the *C-Tree* by taking the tree node as the virtual root node according to the procedure of Algorithm 6.



(a) Facebook r=1 (b) Facebook r=2
(c) Google+ r=1 (d) Google+ r=2
(e) Gowalla r=1 (f) Gowalla r=2
(g) Twitter r=1 (h) Twitter r=2

Fig. 3. Community Size Distribution.

VI. Experimental Study

All experiments have been conducted on a Red Hat Enterprise Linux Server (7.2), with 792GB RAM and Intel(R) Xeon(R) CPU E5-2690 v2 @ 3.00GH. The algorithms are implemented using Python 2.7. The average experimental results of 200 runs with different data inputs at the same settings are reported.

A. Data Sets and Parameter Settings

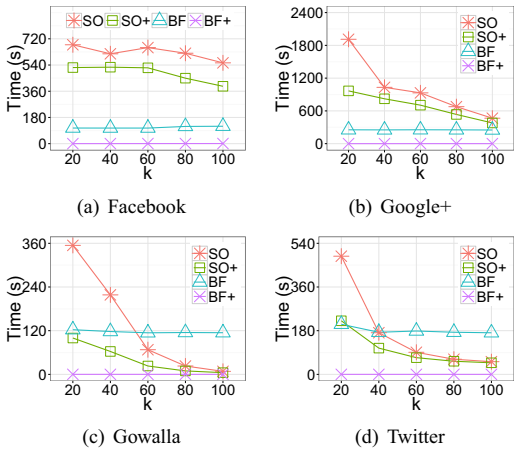
To evaluate the proposed new community model, index and search algorithms, we select six real social network datasets downloaded from Stanford Dataset Collection (https://snap.stanford.edu/data/), as shown in Table I. Figure 3 illustrates the distribution of community size where x-axis is the community size and y-axis indicates the number of communities. For the first four datasets, we have produced three sets of communities for each, denoted as Exp-100, Exp-200 and Exp-300, where each node is constrained to be

Datasets	#nodes	#Edges	Avg Degree
Facebook	4,039	176,468	43.69
Google+	23,628	78,388	3.32
Gowalla	69,097	351,452	5.09
Twitter	486,879	4,293,610	8.82
Youtube	52,675	636,864	12.10
Amazon	317,194	1,745,870	5.50

TABLE I. STATISTICAL INFORMATION OF THE DATASETS

included in at most 100, 200 and 300 communities respectively. When $r = 1$, the size of all communities is less than 80. When $r = 2$, the size of most communities (e.t., approximately 99%) is in between 100 and 500. The proposed methods aim to handle the complex situation where the search space consists of a large number of large communities. So, the experiments focused on the situation when $r = 2$.

B. Efficiency Evaluation of Influential Community Search



(a) Facebook (b) Google+

(c) Gowalla (d) Twitter

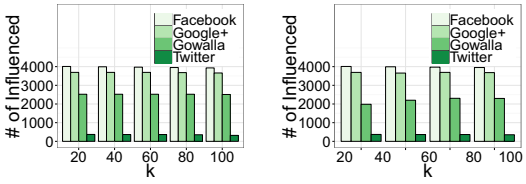
Fig. 4. Search time.

In this subsection, we test the efficiency of the proposed search algorithms (i.e., SO, SO+, BF and BF+) based on the C-tree.

Figure 4 illustrates the experimental results when we vary the value of k as 20, 40, 60, 80, and 100, respectively. When $k = 20$, it means the query only concerns the communities of size no less than 20. In Figure 4, we can see that BF and BF+ are much faster than SO and SO+ on Facebook and Google+ datasets. This is because BF and BF+ first probe the community with the highest upper bound value while SO and SO+ have to probe the communities maintained in *C-Tree* in a sequential order. Thus, BF can prune lots of communities with the smaller upper bound values. Another reason is that Facebook and Google+ are dense datasets and there are many user-to-user edges with higher bi-directed influence, in which SO+ consumes much more time on the influence computation of candidate communities, but BF only needs to estimate a few communities due to the upper bound rules. Interestingly, BF consumes the similar time as SO+ for Gowalla and Twitter. The main reason is that they are sparse datasets and contain less communities. Therefore, SO+ can quickly scan the communities in the *C-Tree* and identify the most influential community. But BF has to spend lots of extra time on sorting the communities, which leads to the lower efficiency.

As the improved BF, BF+ pre-sorts the communities in the *C-Tree* by their upper bounds where each community is represented by a *C-Tree* node ID. For each ID, we online obtain its corresponding community by accessing its subtree and its ancestor nodes in the *C-Tree*. As a result, we have a sorted ID list to guide the search of the most influential community. The test results show that BF+ is much faster than BF in most cases. In particular, when $k = 20$, the time consumed by BF is about 622 times of that by BF+ for Facebook, 28 times for Twitter data, and 5.7-8.2 times for Google+ and Gowalla.

C. Evaluation of Community Influence Spread



(a) Normal Query (b) Range Query

Fig. 5. Influence Scores over Different Datasets.

In this subsection, we conduct two studies on the influence spread of returned influential communities. The first study is to report the influence of the most influential community whose size is not less than a given parameter k (i.e., $\geq k$). The second study is to report the influence of the most influential community whose size is in a range $[k_1, k_2)$ (i.e., $\geq k_1$ and $< k_2$). In Figure 5, the experimental results are presented when k is 20, 40, 60, 80, or 100 in (a) and the results for four ranges [20,40), [40,60), [60,80) and [80,100] are presented in (b). Surprisingly, we notice that the numbers of nodes influenced in different settings of k and $[k_1, k_2)$ are almost same. It shows that the most influential community can influence 3932-4010 persons in Facebook, 3661-3693 persons in Google+, 2509-2522 persons in Gowalla, and 320-371 persons in Twitter. This result delivers two important messages: (**1**) the influence spread of a community is very sensitive to the types of social networks; and (**2**) the communities of smaller size can have the similar influence spread as the ones of larger size.

D. Evaluation of maximal kr-clique Community Model

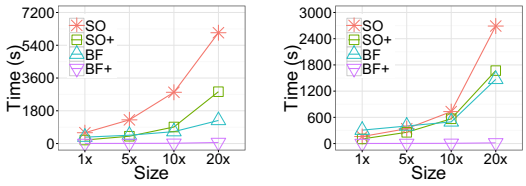
The recent study in [25] comprehensively assessed eight community detection models over different types of datasets. It concluded that LPA (Label Propagation Algorithms) is the most reliable model in generating the desirable communities. In this subsection, we aim to verify the robustness of the maximal kr-clique community model. For this purpose, we identify communities using LPA in the same datasets and treat those communities as the desirable communities. To check to which extent the maximal kr-clique communities are desirable, we compared our maximal kr-clique communities against the LPA generated communities using the metric of *Normalized Mutual Information (NMI)*. The score of NMI stands for the

agreement of two sets of results and is defined as

$$NMI = \frac{-2\sum_{i,j} N_{ij} log \frac{N_{ij} N_t}{N_{i*} N_{*j}}}{\sum_i N_{i*} log \frac{N_{i*}}{N_t} + \sum_j N_{*j} log \frac{N_{*j}}{N_t}}$$

N is the confusion matrix whose element N_{ij} is the number of the shared members between a maximal kr-clique community C_i and a LPA community C_j. N_{i*} and N_{*j} are the sum over row i and column j respectively, and $N_t = \sum_i \sum_j N_{ij}$. The experimental results show that NMI score can achieve about 99% for Twitter, 89% for Gowalla, 70% for Facebook and 63% for Google+. The higher scores in this experiment help to verify that our proposed maximal kr-clique community model is reliable and desirable in practice. We evaluate the robustness of the maximal kr-clique community model using the first four datasets, instead of Amazon and Youtube because Amazon and Youtube only provide the ground truth communities with small size, e.g., the community size is around 13.5 in Youtube. But in this work, we are interested in searching influential communities with large community size between 100 and 500.

E. Efficiency Evaluation of Scalability



(a) k=50 (b) k=100

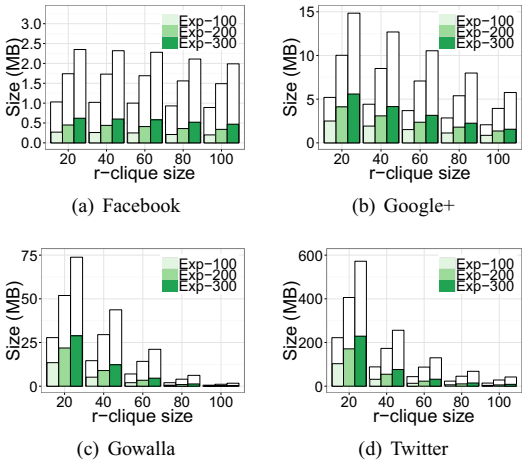
Fig. 6. Scalability Evaluation.

To further evaluate the performance of our proposed index and algorithms, We test the time cost when we increase the size of the Twitter dataset by 5 times, 10 times and 20 times, as shown in Figure 6. When k is 50, the time cost of SO increased linearly. But the increasing trend lines of the other three algorithms grow slowly. The three algorithms SO, SO+ and BF perform similarly when k is set as 100 and the data size is 1 time, 5 times and 10 times, respectively. But when the data size is 20 times, SO+ and BF perform much better than SO. In all settings, BF+ perform the best. This is because BF+ is able to filter out lots of time-consuming community identification and influence spread evaluations.

F. Space and Time Cost Evaluation of Building C-Tree

Given a social network, the minimal duplicate community tree (*C-Tree*) is generated for maintaining communities. The key optimization to *C-Tree* is to reduce the duplicates of nodes appearing in different communities. As discussed in section V, *C-Tree* is constructed using a greedy algorithm which processes nodes in the order of their duplicate level. For a node at the higher duplicate level, it means it is a member in a large number of communities. *C-Tree* is then optimized by checking the nodes at the relatively lower duplicate level but they can further reduce duplicate nodes in *C-Tree*.

Figure 7 compares the space cost of maintaining communities with *C-Tree* and without *C-Tree*. The x-axis indicates the maximum r-clique size that a *C-Tree* supports, i.e., only

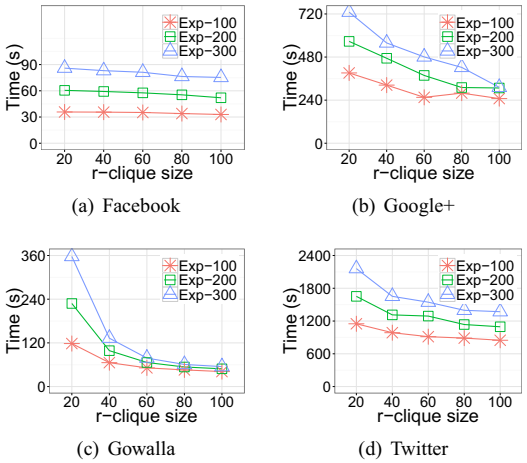


(a) Facebook (b) Google+

(c) Gowalla (d) Twitter

Fig. 7. Space Cost of Buiding C-tree.

the communities containing more than r-clique size of nodes are retrievable from the *C-Tree*; y-axis indicates the storage requirement with *C-Tree* (represented with color section of each bar) and without *C-Tree* (represented with the entire bar). In Figure 7, we observe that the space cost can be reduced by about 75% for Facebook, 65% for Google+, 66% for Gowalla and 65% for Twitter using *C-Tree*. Clearly, the space cost decreases when the size of each maintained r-clique increases because the quantity of communities decreases. Similarly, the quantity of communities decreases from Exp-300 to Exp-100 and thus the space cost decreases from Exp-300 to Exp-100. Less space cost means less computation time is required in community search. Figure 8 reports the time of building a *C-Tree* where the presented time cost does not count the time of enumerating maximum r-cliques. When k_{tree} increases from 20 to 100 for a given social network, it consumes less time to construct *C-Tree*.



(a) Facebook (b) Google+

(c) Gowalla (d) Twitter

Fig. 8. Timoe Cost of Constructing C-tree.

G. Additional Evaluation

In this subsection, we tested the outer influence spread of communities by using the Youtube and Amazon datasets

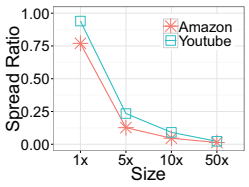


Fig. 9. Influential Ratio in Ground truth Community Datasets.

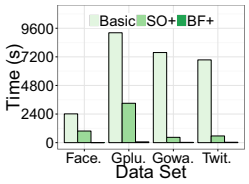


Fig. 10. Time Cost of Basic vs. Index Methods.

because they already provided the ground truth communities in the datasets. We demonstrate the evaluation by testing that how much percentage the ground truth communities can influence the outer node set exceeding their own community sizes by 1 times (1x), 5 times (5x), 10 times (10x) and 50 times (50x). The majority of the communities can influence their outer nodes between 1 time and 5 times. There are 5%-12% communities that can influence 5-10 times of their community sizes. Only a few communities can influence up to 50 times. For SO and SO+, they have to check every maximal kr-clique community, but for BF and BF+, they only check a few communities. This is the reason that BF and BF+ performed much better than SO and SO+.

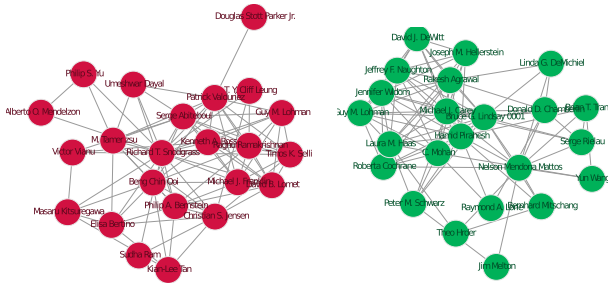
Since the baseline algorithm (denoted as *basic*) needs to calculate the communities, obviously it is much slower than the other algorithms. To verify this, we compare the time cost of the *basic* algorithm with that of SO+ and BF+ over the four real datasets when k is 20. As shown in Figure 10, the *basic* algorithm is slower than SO+ by about 5-7 times on Gowalla and Twitter, about 1.5 times on Google+ and Facebook. This is mainly because Google+ and Facebook have more edges with high bi-directed influence probability. In this situation, SO+ also needs to consume much time on the influence computation. As such, the acceleration of SO+ is just 1.5 times compared with the *basic* algorithm. In all datasets, BF+ outperforms the other two algorithms greatly, which is faster than the *basic* algorithm by about 200 times and SO+ by about 50-100 times.

H. Case Study

We conduct a case study based on the co-authorship network and citation network in database research areas[1]. Our study focuses on the 19,853 papers published by 22,250 authors in the top-10 DB conferences/journals (i.e., SIGMOD, VLDB, PVLDB, ICDE, EDBT, CIKM, TODS, VLDB J., SIGMOD Rec., and TKDE). The citations in 4,943 papers involved 4,558 authors. The influence of a researcher to another one is learned by the citation relationships and the order of their names appearing in the cited papers.

Figure 11 presents the top-2 most influential communities where k=2 and r=2. The results illustrates that Philip S. Yu and Beng Chin Ooi have strong relation via M. Tamer Zsu, which cannot be detected by k-truss. The small case study verifies that the identified most influential communities can detect the collaborative groups with the well recognized influence in the real world. We highlight the top-six contributors in each community. For instance, Philip A. Bernstein can

[1]http://dblp.uni-trier.de/xml/



(a) Top-1 (b) Top-2

Fig. 11. The top 2 influential co-authoring groups.

independently influence 621 authors and Umeshwar Dayal can independently influence 515 authors, which brings benefit to Top-1 community; David J. DeWitt can independently influence 734 and Michael J. Carey can independently influence 722, which brings benefit to Top-2 community.

VII. Related Work

Social Community Detection: A great deal of work has been devoted to find communities in large networks, and much of this has been devoted to formalize the intuition that a community is a set of nodes that has more and/or better links between its members than with the remainder of the network. To design effective community discovery models, Newman and Girvan in [20] proposed a quantitative measure, called modularity, to assess the quality of community structures, and formulated community discovery as an optimization problem. The key idea is similar to graph partitioning, which iteratively removes the edge with the highest betweenness score. Betweenness based community detection metric was also studied by Girvan and Newman in [10]. Ruan and Zhang [22] proposed a more efficient spectral algorithm to find high quality communities by applying k-way partitioning and recursive 2-way partitioning strategies. Satuluri and Parthasarathy in [23] developed efficient Markov clustering algorithms to identify communities by using stochastic flow technique. The key idea is to enhance flow to well-connected nodes, i.e., rich get richer and poor get poorer. All the above implicit community detection models focused on the global connectivity of the social network, by which a specified number of partitions (communities) are generated. The discovered communities often have small cohesiveness. Another line of work is to discover communities based on explicit community model like k-core [18] and k-truss [12]. The k-core of a graph is the largest subgraph within which each node has at least k connections. In the induced subgraph (i.e., a k-core community), since it only requires each node has k neighbors, two nodes may have large hops (i.e., less cohesive). Given a graph G, the k-truss is the largest subgraph in which every edge is contained in at least (k-2) triangles within the subgraph. The k-truss is a type of cohesive subgraph defined based on triangle which models the stable relationship among three nodes. With edge connectivity constraints, the induced subgraph (i.e., a k-truss community) is connected and cohesive. But the connectivity is so strong that k-truss can only be used to discover communities of very small size (i.e., not a society). In this work, we propose a new explicit community model, called maximal *kr*-Clique

community, which is based on the concept of maximal cliques [1]. Compared to k-core and k-truss, the maximal *kr*-Clique community model concurrently has more desirable properties including *society*, *cohesiveness*, *connectivity*, and *maximum*.

Influence Maximization: Kempe et al. [13] proposed two discrete influence propagation models, Independent Cascade (IC) model and Linear Thresholds model. Based on the two widely-accepted models, there are lots of work focusing on influence maximization problem, e.g., [2], [3], [7], [17], [19]. Their common target is to select a limited number of nodes from a social network as the seed nodes such that, at the end of influence propagation process using IC model or LT model, the influence of any information initialized by the seed nodes in the social network is maximized. There is no any constraint on the relationship between the seed nodes. It is very likely for the users represented by the seed nodes who don't know the existence of the others. Besides this, in this paper we are interested in a different research problem, i.e., studying the outer influence of maximal *kr*-Clique communities where the member nodes of the community are closely related. In addition, our problem focuses on the influence to external nodes, excluding its influence to its own members. But this is not concerned by the influence maximization problem.

VIII. Conclusions

This work, for the first time, investigated the problem of the most influential community search in large social networks. In particular, we defined an innovative *maximal kr-cliques community* which possesses more desirable properties compared to the existing community models, such as k-core and k-truss. The maximal kr-cliques community with the highest outer influence has particular interest in many applications of the real world since it provides a new and pragmatic view to interpret the value of social network communities. To enable the most influential community search in large social networks online, the tailored index *C-Tree* and a number of robust search algorithms have been developed. Their efficiency have been verified using six real world social networks. It is worth to indicate that *C-Tree* and the search algorithms can be applied to other community models with straightforward extension. From the broader perspective, this work sheds light to community detection when additional aspects of community are concerned beyond structural constraints.

IX. Acknowledgments

This work is supported by the ARC Discovery Project under grant No. DP160102114. It is also partially supported by the Research Grants Council of Hong Kong SAR, China, under grant No. 14209314 and No. 14221716, as well as the NSF of China under grant No. 61322208 and No. 61532021.

References

[1] C. Bron and J. Kerbosch. Finding all cliques of an undirected graph (algorithm 457). *Commun. ACM*, 16(9):575–576, 1973.

[2] S. Chen, J. Fan, G. Li, J. Feng, K. Tan, and J. Tang. Online topic-aware influence maximization. *PVLDB*, 8(6):666–677, 2015.

[3] W. Chen, T. Lin, and C. Yang. Efficient topic-aware influence maximization using preprocessing. *CoRR*, abs/1403.0057, 2014.

[4] W. Chen, C. Wang, and Y. Wang. Scalable influence maximization for prevalent viral marketing in large-scale social networks. In *SIGKDD*, pages 1029–1038, 2010.

[5] W. Chen, Y. Wang, and S. Yang. Efficient influence maximization in social networks. In *SIGKDD*, pages 199–208, 2009.

[6] Y. Chen, W. Peng, and S. Lee. Efficient algorithms for influence maximization in social networks. *Knowl. Inf. Syst.*, 33(3):577–601, 2012.

[7] P. M. Domingos and M. Richardson. Mining the network value of customers. In *SIGKDD*, pages 57–66, 2001.

[8] D. Eppstein, M. Löffler, and D. Strash. Listing all maximal cliques in sparse graphs in near-optimal time. In *Algorithms and Computation - Part I*, pages 403–414, 2010.

[9] U. Feige. Approximating maximum clique by removing subgraphs. *SIAM J. Discrete Math.*, 18(2):219–225, 2004.

[10] M. Girvan and M. Newman. Community structure in social and biological networks. *PNAS*, 99(12):7821–7826, 2002.

[11] A. Goyal, F. Bonchi, and L. V. S. Lakshmanan. Learning influence probabilities in social networks. In *WSDM*, pages 241–250, 2010.

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

[13] D. Kempe, J. M. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *SIGKDD*, pages 137–146, 2003.

[14] M. Kimura and K. Saito. Tractable models for information diffusion in social networks. In *PKDD*, pages 259–271, 2006.

[15] J. Lee and C. Chung. A query approach for influence maximization on specific users in social networks. *IEEE Trans. Knowl. Data Eng.*, 27(2):340–353, 2015.

[16] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. M. VanBriesen, and N. S. Glance. Cost-effective outbreak detection in networks. In *SIGKDD*, pages 420–429, 2007.

[17] J. Li, C. Liu, J. X. Yu, Y. Chen, T. K. Sellis, and J. S. Culpepper. Personalized influential topic search via social network summarization. *IEEE Trans. Knowl. Data Eng.*, 28(7):1820–1834, 2016.

[18] R. Li, J. X. Yu, and R. Mao. Efficient core maintenance in large dynamic graphs. *IEEE Trans. Knowl. Data Eng.*, 26(10):2453–2465, 2014.

[19] Y. Li, D. Zhang, and K. Tan. Real-time targeted influence maximization for online advertisements. *PVLDB*, 8(10):1070–1081, 2015.

[20] M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69(026113), 2004.

[21] S. Qiu, J. Cheng, X. Zhang, B. Niu, and H. Lu. Community discovering guided cold-start recommendation: A discriminative approach. In *IEEE ICME*, pages 1–6, 2014.

[22] J. Ruan and W. Zhang. An efficient spectral algorithm for network community discovery and its applications to biological and social networks. In *IEEE ICDM*, pages 643–648, 2007.

[23] V. Satuluri and S. Parthasarathy. Scalable graph clustering using stochastic flows: applications to community discovery. In *ACM SIGKDD*, pages 737–746, 2009.

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

[25] M. Wang, C. Wang, J. X. Yu, and J. Zhang. Community detection in social networks: An in-depth benchmarking study with a procedure-oriented framework. *PVLDB*, 8(10):998–1009, 2015.