Reverse Influential Community Search Over Social Networks (Technical Report)

Qi Wen
East China Normal University
Shanghai, China
51265902057@stu.ecnu.edu.cn

Nan Zhang
East China Normal University
Shanghai, China
51255902058@stu.ecnu.edu.cn

Yutong Ye
East China Normal University
Shanghai, China
52205902007@stu.ecnu.edu.cn

Xiang Lian
Kent State University
Kent, Ohio, USA
xlian@kent.edu

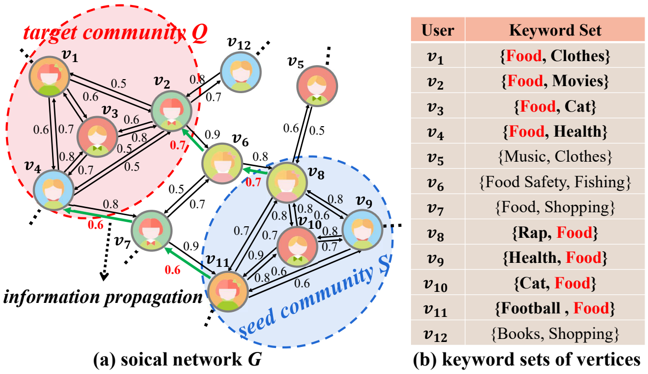
Mingsong Chen
East China Normal University
Shanghai, China
mschen@sei.ecnu.edu.cn

ABSTRACT

As an important fundamental task of numerous real-world applications such as social network analysis and online advertising/marketing, several prior works studied influential community search, which retrieves a community with high structural cohesiveness and maximum influences on other users in social networks. However, previous works usually considered the influences of the community on *arbitrary* users in social networks, rather than specific groups (e.g., customer groups, or senior communities). Inspired by this, we propose a novel ***R**everse **I**nfluential **C**ommunity **S**earch* (RICS) problem, which obtains a *seed community* with the maximum influence on a user-specified *target community*, satisfying both structural and keyword constraints. To efficiently tackle the RICS problem, we design effective pruning strategies to filter out false alarms of candidate seed communities, and propose an effective index mechanism to facilitate the community retrieval. We also formulate and tackle an RICS variant, named ***R**elaxed **R**everse **I**nfluential **C**ommunity **S**earch* (R^2ICS), which returns a subgraph with the relaxed structural constraints and having the maximum influence on a user-specified target community. Comprehensive experiments have been conducted to verify the efficiency and effectiveness of our RICS and R^2ICS approaches on both real-world and synthetic social networks under various parameter settings.

PVLDB Reference Format:
Qi Wen, Nan Zhang, Yutong Ye, Xiang Lian, and Mingsong Chen. Reverse Influential Community Search Over Social Networks (Technical Report). PVLDB, 18(1): XXX-XXX, 2025.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/Luminous-wq/RICS-Reverse-Influential-Community-Search.



(a) soical network G

User	Keyword Set
v_1	{Food, Clothes}
v_2	{Food, Movies}
v_3	{Food, Cat}
v_4	{Food, Health}
v_5	{Music, Clothes}
v_6	{Food Safety, Fishing}
v_7	{Food, Shopping}
v_8	{Rap, Food}
v_9	{Health, Food}
v_{10}	{Cat, Food}
v_{11}	{Football , Food}
v_{12}	{Books, Shopping}

(b) keyword sets of vertices

Figure 1: An RICS example over social network G.

1 INTRODUCTION

For the past decades, the *community search* has attracted much attention in various real-world applications such as online advertising/marketing [1–4], social network analysis [5–8], and many others. Prior works on the community search [9–12] usually retrieved a community (subgraph) of users from social networks with high structural and/or spatial cohesiveness. Several existing works [11, 13, 14] considered the influences of communities and studied the problem of finding communities with high influences on other users in social networks.

In this paper, we propose a novel problem, named *Reverse Influential Community Search* (RICS) over social networks, which obtains a community (w.r.t. specific interests such as sports, food, etc.) that has high structural cohesiveness and the highest influence on a targeted group (community) of users (instead of arbitrary users in social networks). The resulting RICS communities are useful for various real applications such as online advertising/marketing in social media [15] and disease spread prevention in contact networks [16]. Below, we give motivation examples of our RICS problem.

Example 1. ***(Online Advertising and Marketing Over Social Networks)*** *In social networks (e.g., Twitter), a sales manager wants to ensure the optimal advertisement dissemination of some products to a targeted group of users through social networks. Figure 1(a) shows an example of social network G, where each user vertex v_i ($1 \le i \le 12$) is associated with a set of keywords (indicating the user's interests, as*

Proceedings of the VLDB Endowment, Vol. 18, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

depicted in Fig. 1(b)). For example, user v_3 is interested in delicious food and cute cats. In this scenario, the sales manager can specify a group of targeted customers (e.g., $v_1 \sim v_4$) for online advertising and marketing (forming a target community Q), and issue an RICS query to identify a seed community, S, of users who have the highest impact on the targeted customers in Q (e.g., via tweets/retweets in Twitter). Users in the returned seed community S will be given coupons or discounts to promote the products on social networks, and most importantly, indirectly affect the targeted customers' purchase decisions. ■

Example 2. ***(Disease Spread Prevention via Contact Networks)*** *In real application of infectious disease prevention, there exists some community of vulnerable people (e.g., senior/minor people) who are either reluctant or unable to take preventive actions such as vaccines, due to religion, age, and/or health reasons. The health department may want to identify a group of people (e.g., relatives, or colleagues) through contact networks [16] who are most likely to spread infectious diseases to such a vulnerable community, and persuade them to use preventive means (e.g., COVID-19 vaccine). In this case, the health department can exactly perform an RICS query to obtain a seed community, S, of people who have the highest disease spreading possibilities to the targeted vulnerable community Q.* ■

The RICS problem has many other real applications such as finding a group of researchers with the highest influence on another target research community in bibliographical networks.

Inspired by the examples above, in this paper, we consider the RICS problem, which obtains a community (called *seed community*) that contains query keywords (e.g., food and clothing) and has the highest influence on a target user group. The resulting RICS community contains highly influential users to whom we can promote products for online advertising/marketing, or suggest taking vaccines for protecting vulnerable people in contact networks.

Note that, efficient and effective answering of RICS queries is quite challenging. A straightforward method is to enumerate all possible communities (subgraphs), compute the influence score of each community with respect to the target group, and return a community with the highest influence score. However, this approach is not feasible in practice, due to the large number of candidate communities.

To the best of our knowledge, previous works have not considered the influences on a target user group. Therefore, previous techniques cannot work directly on our RICS problem. To address the challenges of our RICS problem, we propose a two-stage RICS query processing framework in this paper, including offline precomputation and online RICS querying. In particular, we propose effective pruning strategies (w.r.t., query keyword, boundary support, and influence score) to safely filter out invalid candidate seed communities and reduce the RICS problem search space. Furthermore, we design an effective indexing mechanism to integrate our pruning methods seamlessly and develop an efficient algorithm for RICS query processing.

In this paper, we make the following major contributions.

(1) We formally define the *reverse influential community search* (RICS) problem and its variant, *relaxed reverse influential community search* (R^2ICS), on social networks in Section 2.
(2) We design an efficient query processing framework for answering RICS queries in Section 3.
(3) We propose effective pruning strategies to reduce the RICS problem search space in Section 4.
(4) We devise offline pre-computation and indexing mechanisms in Section 5 to facilitate pruning and online RICS algorithms in Section 6.
(5) We develop an efficient online R^2ICS processing algorithm to retrieve community answers with the relaxed constraints in Section 7.
(6) We demonstrate through extensive experiments the effectiveness and efficiency of our RICS/R^2ICS query processing algorithms over real/synthetic graphs in Section 8.

2 PROBLEM DEFINITION

This section first gives the data model for social networks with the information propagation in Section 2.1, then provides the definitions of target and seed communities in social networks in Section 2.2, and finally formulate a novel problem of *Reverse Influential Community Search* (RICS) over social networks in Section 2.3.

2.1 Social Networks

In this subsection, we model social networks by a graph below.

Definition 1. *(**Social Network**, G) A social network G is a connected graph in the form of a triple $(V(G), E(G), \Phi(G))$, where $V(G)$ and $E(G)$ represent the sets of vertices (users) and edges (relationships between users) in the graph G, respectively, and $\Phi(G)$ is a mapping function: $V(G) \times V(G) \rightarrow E(G)$. Each vertex $v_i \in V(G)$ has a keyword set $v_i.L$, and each edge $e_{u,v} \in E(G)$ is associated with an activation probability $P_{u,v}$.*

In a social-network graph G (given by Definition 1), each user vertex v_i contains topic keywords (e.g., user-interested topics like movies and sports) in a set $v_i.L$, and each edge $e_{u,v}$ is associated with an activation probability, $P_{u,v}$, which indicates the influence from user u to user v through edge $e_{u,v}$. Here, the activation probability, $P_{u,v}$, can be obtained based on node attributes (e.g., interests, trustworthiness, locations) [17], network topology (e.g., node degree, connectivity) [18, 19], or machine learning techniques [15].

Information Propagation Model: In social networks G, we consider an information propagation model defined below.

Definition 2. *(**Information Propagation Model**) Given an acyclic path $Path_{u,v} = u_1 \rightarrow u_2 \rightarrow \cdots \rightarrow u_m$ between vertices u ($= u_1$) and v ($= u_m$) in the social network G, we define the influence propagation probability, $Pr(Path_{u,v})$, from u to v as:*

$$Pr(Path_{u,v}) = \prod_{i=1}^{m-1} P_{u_i,u_{i+1}}, \tag{1}$$

where $P_{u_i,u_{i+1}}$ is the activation probability from vertex u_i to vertex u_{i+1}.

Following the *maximum influence path* (MIP) model [20], an MIP, $MIP_{u,v}$, is a path from u to v with the highest influence propagation probability (among all paths $Path_{u,v}$), which is:

$$MIP_{u,v} = \arg\max_{\forall Path_{u,v}} Pr(Path_{u,v}). \tag{2}$$

The *influence score*, $inf\_score_{u,v}$, from vertex u to vertex v in the social network G is given by:

$$inf\_score_{u,v} = Pr(MIP_{u,v}). \quad (3)$$

2.2 Community

In this subsection, we formally define two terms, *target* and *seed community*, as well as the influence from a seed community to a target community, which will be used for formulating our RICS problem.

Target Community: A *target community* is a group of users whom we would like to influence. For example, in the real application of online advertising/marketing, the target community contains the targeted customers to whom we would like to promote some products; for disease prevention, the target community may contain vulnerable people (e.g., senior/minor people) whom we want to protect from infectious diseases.

Formally, we define the target community as follows.

Definition 3. ***(Target Community)*** *Given a social network G, a center vertex v_q, a list, L_q, of query keywords, and the maximum radius r, a target community, Q, is a connected subgraph of G (denoted as $Q \subseteq G$), such that:*

- *$v_q \in V(Q)$;*
- *for any vertex $v_i \in V(Q)$, we have $dist(v_q, v_i) \leq r$, and;*
- *for any vertex $v_i \in V(Q)$, its keyword set $v_i.L$ contains at least one query keyword in L_q (i.e., $v_i.L \cap L_q \neq \emptyset$),*

where $dist(x, y)$ is the shortest path distance between x and y in Q.

[wrong place? it is for seed community, not target community]

To get a high-influence seed community, the seed community should maintain a highly connected structure. Therefore, we define seed community based on the k-truss structure [21, 22]. The k-truss subgraph requires that two vertices of each edge in the subgraph have at least k-2 common neighbors, which intuitively means that the count of triangles on each edge of the k-truss subgraph should be greater than k-2. The k-truss subgraph structure based on *triangle* constraints better represents the dense structure of the subgraph.

Seed Community: In addition to directly influence the target community (e.g., advertising to targeted users in social networks, or protecting vulnerable people in contact networks), we can also find a group of other users in G (for advertising or protecting, resp.) that indirectly and highly influence the target community. Such a group of influential users forms a *seed community*, as defined below.

Definition 4. ***(Seed Community)*** *Given a social network G, a set, L_q, of query keywords, a center vertex v_s, an integer parameter k, the maximum number, N, of community users, and the maximum radius r, a seed community, S, is a connected subgraph of G (denoted as $S \subseteq G$), such that:*

- *$v_s \in V(S)$;*
- *$|V(S)| \leq N$;*
- *S is a k-truss [21];*
- *for any vertex $v_i \in V(S)$, we have $dist(v_s, v_i) \leq r$, and;*
- *for any vertex $v_i \in V(S)$, its keyword set $v_i.L$ contains at least one query keyword in L_q (i.e., $v_i.L \cap L_q \neq \emptyset$).*

The Calculation of the Community-to-Community Influence: We next define the community-level influence, $inf\_score_{S,Q}$, from a seed community S to a target community Q (w.r.t. topic keywords in L_q) in social networks G.

Definition 5. ***(Community-to-Community Influence)*** *Given a target community Q, a seed community S, the community-to-community influence, $inf\_score_{Q,S}$, of seed community S on target community Q is defined as:*

$$inf\_score_{S,Q} = \sum_{u \in V(S)} \sum_{v \in V(Q)} inf\_score_{u,v}, \quad (4)$$

where $inf\_score_{u,v}$ is the influence of vertex u on vertex v (as given in Equation (3)).

Intuitively, in Definition 5, the community-to-community influence $inf\_score_{S,Q}$ (as given in Equation (4)) calculates the summed influence for all user pairs (in other words, collaborative influence from users in seed community S to that in target community Q).

2.3 The Problem Definition of Reverse Influential Community Search Over Social Networks

In this subsection, we propose a novel problem, named *Reverse Influential Community Search* (RICS) over social networks, which retrieves a seed community S with the highest influence on a given target community in a social network G.

The RICS Problem Definition: Formally, we have the following RICS problem definition.

Definition 6. ***(Reverse Influential Community Search Over Social Networks, RICS)*** *Given a social network $G = (V(G), E(G), \Phi(G))$, a set, L_q, of query keywords, an integer parameter k, the maximum number, N, of community users, and a target community Q (with center vertex v_q, radius r, and query keywords in L_q), the problem of reverse influential community search (RICS) returns a connected subgraph (community), S, from the social network G, such that:*

- *S satisfies the constraints of a seed community (as given in Definition 4), and;*
- *the community-to-community influence, $inf\_score_{S,Q}$, is maximized (i.e., $S = \arg\max_{S \subseteq G} inf\_score_{S,Q}$).*

Intuitively, the RICS problem retrieves a keyword-aware seed community S that has the highest influence on the target community Q. In real applications such as online advertising/marketing, we can issue the RICS query over the social network G and obtain a seed community S of users to whom we can give group buying coupons or discounts to (indirectly) influence the targeted customers in the target community Q.

A Variant, R^{2}ICS, of the RICS Problem: In Definition 6, our RICS problem will return the maximum influential seed community. Note that, in Definition 4, very few communities will fulfill the structural requirements of a seed community in the real world. To generalize the RICS problem, in this paper, we also consider a variant of the RICS, named ***R****elaxed* ***R****everse* ***I****nfluential* ***C****ommunity* ***S****earch* (R^{2}ICS), which obtains a set of generalized community with the highest influence.

Table 1: Symbols and Descriptions

Symbol	Description
G	a social network
$V(G)$	a set of vertices v_i
$E(G)$	a set of edges $e(u, v)$
$\Phi(G)$	a mapping function $V(G) \times V(G) \rightarrow E(G)$
S (or Q)	a seed community (or target community) in G
L_q	a set of query keywords
$v_i.L$	a set of keywords associated with user v_i
$v_i.BV$	a bit vector with the hashed keywords in $v_i.L$
$Path_{u,v}$	an acyclic path from user u to user v
$Pr(Path_{u,v})$	the propagation probability that user u activates user v through an acyclic path $Path_{u,v}$
$inf\_score_{u,v}$	the influence score of vertex u on vertex v
$inf\_score_{S,Q}$	the community-to-community influence of S on Q
r-$hop(v_i, G)$	a subgraph in G with v_i as the vertex and r as the radius
r	the user-specified radius of target and seed communities
k	the support parameter in k-truss for the seed community
$sup(e_{u,v})$	the support of edge $e_{u,v}$
θ	the influence threshold

Definition 7. ***(Relaxed Reverse Influential Community Search, R^{2}ICS)*** *Given a social network $G = (V(G), E(G), \Phi(G))$, a set, L_q, of query keywords, the maximum number, N, of community users, and a target community Q (with center vertex v_q, radius r, and query keywords in L_q), the problem of the relaxed reverse influential community search (R^{2}ICS) returns a subgraph, S, from the social network, G, such that:*

- *S is a subgraph of G with size $|V(S)| \leq N$,*
- *for any vertex $v_i \in V(S)$, its keyword set $v_i.L$ contains at least one query keyword in L_q (i.e., $v_i.L \cap L_q \neq \emptyset$),and;*
- *the community-to-community influence, $inf\_score_{S,Q}$, is maximized (i.e., $S = \arg\max_{S \subseteq G} inf\_score_{S,Q}$).*

Different from retrieving the seed community in the RICS problem (as given in Definition 6), the variant, R^{2}ICS, in Definition 7 returns a subgraph community S without the structural constraints such as k-truss and radius r.

Table 1 lists the commonly used notations and their descriptions in this paper.

3 THE RICS FRAMEWORK

Algorithm 1 presents our framework for efficiently processing the RICS query, which consists of two phases, that is, *offline pre-computation* and *online RICS-computation* phases.

During the offline pre-computation phase, we pre-calculate some data from social networks (for effective pruning) and construct an index over the pre-computed data, which can be used for subsequent online RICS processing. Specifically, for each vertex v_i in the social network G, we first hash its set, $v_i.L$, of keywords into a bit vector $v_i.BV$ (lines 1-2). We also pre-calculate a distance vector, $v_i.Dist$, which stores the shortest path distances from vertex v_i to pivots $piv \in S_{piv}$, where S_{piv} is a set of d carefully selected vertices (line 3). Next, we pre-compute the support bounds, boundary influence upper bound, and influence set for r-hop subgraphs (centered at vertex v_i and with radii r ranging from 1 to r_{max}), in order to facilitate the pruning (lines 4-7). Afterward, we construct a tree index $\mathcal{I}$ on the pre-computed data (line 8).

During the online RICS-computation phase, for each user-specified RICS query, we traverse the index $\mathcal{I}$ and apply our proposed pruning strategies (w.r.t. keywords, support, and influence score) to obtain candidate seed communities (lines 9-10). Finally, we calculate the influence scores between candidate seed communities and Q to obtain the best seed community (line 11).

Algorithm 1: The RICS Processing Framework

```
Input: i) a social network $G$, ii) a set, $L_q$, of query keywords, iii) the
       maximum radius, $r$, of each community, iv) an integer parameter, $k$,
       of the $k$-truss, v) the maximum user number, $N$, for each seed
       community, vi) the query center vertex $v_q$, and vii) a set, $S_{piv}$, of
       pivots
Output: a seed community, $S$, with the highest influential score
// offline pre-computation phase
1  for each $v_i \in V(G)$ do
2      hash keywords in $v_i.L$ into a bit vector $v_i.BV$
3      compute a vector, $v_i.Dist$, of distances from $v_i$ to all pivots $piv \in S_{piv}$
4      for $r = 1$ to $r_{max}$ do
5          extract $r$-hop subgraph $r$-$hop(v_i, G)$
6          compute the upper bound of support $ub\_sup(.)$ in $r$-$hop(v_i, G)$
7          compute the upper bound of boundary influence
           $ub\_bound\_inf_r(.)$ in $r$-$hop(v_i, G)$
8  build a tree index $\mathcal{I}$ over graph $G$ with pre-computed data as aggregates
// online RICS-computation phase
9  for each RICS query do
10     traverse the tree index $\mathcal{I}$ by applying keyword, support, and influence
         score pruning strategies to retrieve candidate seed communities
11     calculate the influence scores of candidate seed communities and return
         the one with the highest influential score
```

4 PRUNING STRATEGIES

In this section, we present effective pruning strategies that reduce the problem search space during the online RICS-computation phase (lines 9-11 of Algorithm 1).

4.1 Keyword Pruning

According to Definitions 3 and 4, each vertex in the target/seed community Q or S must contain at least one keyword from the query keyword set L_q. Therefore, our keyword pruning method can filter out those candidate subgraphs that do not meet this criterion.

Lemma 4.1. **(Keyword Pruning)** *Given a set, L_q, of query keywords and a candidate subgraph (community) S, any vertex $v_i \in V(S)$ can be safely pruned from S, if it holds that: $v_i.L \cap L_q = \emptyset$, where $v_i.L$ is the keyword set associated with vertex v_i.*

Proof. If $v_i.L \cap L_q = \emptyset$ holds for any user vertex v_i in a candidate community S, it indicates that user v_i is not interested in any keyword in the query keyword set L_q. Thus, user vertex v_i does not satisfy the keyword constraint in Definition 4, and vertex v_i can be safely pruned from S, which completes the proof. □ □

4.2 Support Pruning

From Definition 4, the seed community S needs to be a k-truss [21]. Denote the support, $sup(e_{u,v})$ of an edge $e_{u,v}$ as the number of triangles containing $e_{u,v}$. Each edge $e_{u,v}$ in the seed community S, is required to have its support $sup(e_{u,v})$ greater than or equal to $(k - 2)$. If we can obtain an upper bound, $ub\_sup(e_{u,v})$, of the support for each edge in the candidate seed community S, then we can employ the following lemma to eliminate candidate seed communities with low support.

Lemma 4.2. **(Support Pruning)** *Given a candidate seed community S and a positive integer k (> 2), an edge $e_{u,v}$ in S can be discarded safely from S, if it holds that $ub\_sup(e_{u,v}) < k - 2$, where $ub\_sup(e_{u,v})$ is an upper bound of the edge support $sup(e_{u,v})$.*

Proof. In the definition of the k-truss [21], the support value, $sup(e_{u,v})$, of the edge $e_{u,v}$ is determined by the number of triangles that contain edge $e_{u,v}$. In a k-truss, each edge must be reinforced by at least $(k - 2)$ such triangle structures. Since we have the conditions that $ub\_sup(e_{u,v}) < k - 2$ (lemma assumption) and $sup(e_{u,v}) \leq ub\_sup(e_{u,v})$ (support upper bound property), by the inequality transition, we have $sup(e_{u,v}) < k - 2$. Therefore, based on Definition 4, the k-truss seed community S cannot include the edge $e_{u,v}$ due to its low support (i.e., $< k - 2$). We thus can safely rule out edge $e_{u,v}$ from S, which completes the proof. □ □

4.3 Influence Score Pruning

In this subsection, we provide an effective pruning method to filter out candidate seed communities with low influence scores.

Since the exact calculation of the influence score between two communities (given by Equation (4)) is very time-consuming, we can take the maximum influence from candidate seed communities that we have seen as an influence score upper bound (denoted as an influence threshold θ). This way, we can apply the influence score pruning in the lemma below to eliminate those seed communities with low influences.

Lemma 4.3. **(Influence Score Pruning)** *Let an influence threshold θ be the maximum influence from candidate seed communities we have obtained so far to the target community Q. Any candidate seed community S can be safely pruned, if it holds that $ub\_inf\_score_{S,Q} < \theta$, where $ub\_inf\_score_{S,Q}$ is an upper bound of the influence score $inf\_score_{S,Q}$ (given by Equation (4)).*

Proof. Since $ub\_inf\_score_{S,Q}$ is an upper bound of the influence score $inf\_score_{S,Q}$, we have $inf\_score_{S,Q} \leq ub\_inf\_score_{S,Q}$. Due to the lemma assumption that $ub\_inf\_score_{S,Q} < \theta$, by the inequality transition, it holds that $inf\_score_{S,Q} < \theta$, which indicates that the candidate community S has lower influence on Q, compared with some communities we have obtained so far (i.e., with influence θ), and S cannot be our RICS answer. Therefore, we can safely prune candidate seed community S, which completes the proof. □ □

5 OFFLINE PRE-COMPUTATION

In this section, we discuss how to offline pre-compute data over social networks, and construct a tree index $\mathcal{I}$ on pre-computed data (lines 1-8 of Algorithm 1).

5.1 Offline Pre-Computed Data

In order to facilitate online RICS computation, we first conduct offline pre-computations on the social network G in Algorithm 2, which can obtain aggregated information about candidate seed communities (later used for pruning strategies to reduce the online search cost). Specifically, for each vertex v_i, we hash a set, $v_i.L$, of its keywords into a bit vector $v_i.BV_0$ of size B, and initialize a pre-computed set, $v_i.Aux$, of auxiliary data with $v_i.BV_0$ (lines 1-3). Then, we compute the distances from v_i to d pivots in S_{piv}, forming a distance vector $v_i.Dist$ of size d, and add $v_i.Dist$ to $v_i.Aux$ (lines 4-5). Next, we compute a support upper bound, $ub\_sup(e_{u,v})$, for each edge $e_{u,v}$ in a subgraph, r_{max}-$hop(v_i, G)$, centered at vertex v_i and with radius r_{max} (lines 6-7). Then, for each vertex v_i and possible radius $r \in [1, r_{max}]$, we pre-compute a keyword bit vector (lines 8-10), an edge support upper bound (line 11), and an upper bound, $v_i.ub\_bound\_inf_r$, of boundary influence scores (line 12) for r-$hop(v_i, G)$ subgraph. Finally, we add these pre-computed aggregated information to $v_i.Aux$ in the following format: $\{v_i.BV_0, v_i.Dist, v_i.BV_r, v_i.ub\_sup_r, v_i.ub\_bound\_inf_r\}$ (line 13).

Algorithm 2: Offline Pre-Computation

```
Input: i) a social network $G$; ii) the maximum radius, $r_{max}$, of each
       community, and; iii) a set, $S_{piv}$, of $d$ pivots
Output: pre-computed auxiliary data $v_i.Aux$ for each vertex $v_i$
1  for each $v_i \in V(G)$ do
       // the keyword bit vector
2      hash keywords in $v_i.L$ into a bit vector $v_i.BV_0$
3      $v_i.Aux = \{v_i.BV_0\}$
       // the distance vector to pivots
4      compute a vector, $v_i.Dist$, of distances from vertex $v_i$ to $d$ pivots in $S_{piv}$
5      add $v_i.Dist$ to $v_i.Aux$
       // edge support upper bounds
6      for each $e_{u,v} \in E(r_{max}\text{-}hop(v_i, G))$ do
7          compute on edge support upper bound $ub\_sup(e_{u,v})$
8  for each $v_i \in V(G)$ do
9      for $r = 1$ to $r_{max}$ do
10         $v_i.BV_r = \bigvee_{\forall v_l \in r\text{-}hop(v_i,G)} v_l.BV$
11         $v_i.ub\_sup_r = \max_{\forall e_{u,v} \in E(r\text{-}hop(v_i,G))} ub\_sup(e_{u,v})$
12         $v_i.ub\_bound\_inf_r = max\{\text{collapse\_calculate}(r\text{-}hop(v_i, G))\}$
13         add $v_i.BV_r$ , $v_i.ub\_sup_r$ and $v_i.ub\_bound\_inf_r$ to $v_i.Aux$
14 return $v_i.Aux$
```

To summarize, $v_i.Aux$ contains the following information:

- **a bit vector, $v_i.BV_0$, of size B**, which is obtained by using a hashing function $f(l)$ to hash each keyword $l \in v_i.L$ to an integer between $[0, B - 1]$ and set the $f(l)$-th bit position to 1 (i.e., $v_i.BV_0[f(l)] = 1$);
- **a distance vector, $v_i.Dist$, of size d**, which is obtained by computing the shortest path distances, $dist(v_i, piv)$, from v_i to d pivots $piv_j \in S_{piv}$; (i.e., $v_i.Dist[j] = dist(v_i, piv_j)$ for $0 \leq j < d$);
- **a bit vector, $v_i.BV_r$ (for $1 \leq r \leq r_{max}$)**, which is obtained by hashing each keyword in keyword set $v_l.L$ of a vertex v_l in the subgraph r-$hop(v_i, G)$ into a position in the bit vector (i.e., $v_i.BV_r = \bigvee_{\forall v_l \in r\text{-}hop(v_i,G)} v_l.BV$);
- **a support upper bound, $v_i.ub\_sup_r$**, which is obtained by taking the maximum of all support bounds $ub\_sup(e_{u,v})$ for edges $e_{u,v}$ in the subgraph r-$hop(v_i, G)$ (i.e., $v_i.ub\_sup_r = \max_{\forall e_{u,v} \in E(r\text{-}hop(v_i,G)))} ub\_sup(e_{u,v})$), and;
- **an upper bound, $v_i.ub\_bound\_inf_r$, of boundary influence scores**, which is obtained by computing the virtual collapse of a subgraph r-$hop(v_i, G)$ discussed below (i.e., $v_i.ub\_bound\_inf_r = max\{\text{collapse\_calculate}(r\text{-}hop(v_i, G))\}$), where function collapse_calculate(·) returns a set, $v_i.BIS$, of influence scores through boundary vertices.

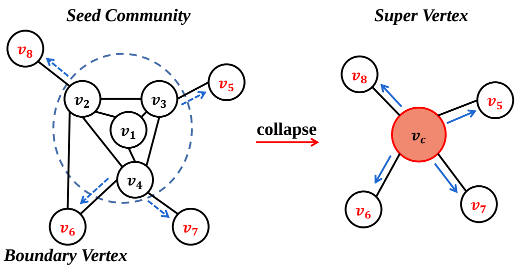


Figure 2: An example of seed community virtual collapse operation. The blue arrows represent the information propagation. v_c represents a virtual super vertex after the community has collapsed. The red vertices represent the boundary vertices.

Discussions on How to Implement collapse_calculate$(\cdot)$**:** Collapse calculations are divided into *target collapse* and *seed collapse*. The difference between the two collapses is that the information propagation is in different directions. As shown in Figure 2, a seed community consisting of v_1, v_2, v_3, and v_4 sends influence to the 1-hop boundary vertices (i.e., v_5, v_6, v_7, and v_8). According to Equation (3) and (4), we aggregate the influence of seed communities towards their external boundary vertices. For a virtual collapsed vertex v_c of a seed community S, through any 1-hop subgraph boundary vertex v_i, we have $inf\_score_{v_c,v_i} = \sum_{v \in V(S)} inf\_score_{v,v_i}$. Finally, we store the set of boundary influence scores $v_i.BIS$ in the center vertex v_i of the community.

Complexity Analysis: As shown in Algorithm 2, for each vertex $v_i \in V(G)$ in the first loop, the time complexity of computing a keyword bit vector $v_i.BV$ is given by $O(|L|)$ (lines 2-3). And the time complexity of computing a distance vector $v_i.Dist$ is given by $O((|V(G)| + |E(G)|) \cdot log|V(G)|)$ (lines 4-5). Let avg_deg denote the average number of vertex degrees. Since there are $avg\_deg^{r_{max}}$ edges in r_{max}-$hop(v_i, G)$ and the cost of the support upper bound computation is a constant (counting the common neighbors), so the time cost of obtaining all edge support upper bounds is $O(avg\_deg^{r_{max}})$ (lines 6-7). Thus, the complexity of the first loop (lines 1-7) is given by $O(|V(G)| \cdot (|L| + (|V(G)| + |E(G)|) \cdot log|V(G)| + avg\_deg^{r_{max}}))$.

In the second loop, for each $v_i \in V(G)$, there are avg_deg^{r-1} vertices in the r-$hop(v_i, G)$ w.r.t. r. Then, for each $r \in [1, r_{max}]$, the time complexity of computing $v_i.BV_r$ and $v_i.ub\_sup_r$ is given by $O(B \cdot avg\_deg^{r-1})$ and $O(avg\_deg^{r-1})$, respectively (lines 10-11). As described in Section 5.1, the time complexity of collapse_calculate$(\cdot)$ is $O((avg\_deg^r + avg\_deg^{r-1}) \cdot log(avg\_deg^{r-1}))$, and so, the time complexity of $v_i.ub\_bound\_inf_r$ is given by $O(n \cdot (avg\_deg^r + avg\_deg^{r-1}) \cdot log(avg\_deg^{r-1}))$. Therefore, the time cost of the second loop (lines 8-13) is $O(|V(G)| \cdot r_{max} \cdot (B \cdot avg\_deg^{r-1} + avg\_deg^{r-1} + n \cdot (avg\_deg^r + avg\_deg^{r-1}) \cdot log(avg\_deg^{r-1})))$.

In summary, the total time complexity of the total offline precomputation is given by $O(|V(G)| \cdot (|L| + (|V(G)| + |E(G)|) \cdot log|V(G)| + avg\_deg^{r_{max}} + r_{max} \cdot (B \cdot avg\_deg^{r-1} + avg\_deg^{r-1} + n \cdot (avg\_deg^r + avg\_deg^{r-1}) \cdot log(avg\_deg^{r-1}))))$.

5.2 Indexing Mechanism

In this subsection, we show the details of offline construction of a tree index $\mathcal{I}$ on a social network G to support online RICS query processing.

The Data Structure of Index $\mathcal{I}$: We will build a tree index $\mathcal{I}$ on the social network G, where each index node, $\mathcal{N}$ includes multiple entries $\mathcal{N}_i$, each corresponding to a subgraph of G. Specifically, the tree index $\mathcal{I}$ contains two types of nodes, leaf and non-leaf nodes. *Leaf Nodes:* Each leaf node $\mathcal{N}$ contains multiple vertices v_i in the corresponding subgraph. The community subgraph centered at v_i is denoted by r-$hop(v_i, G)$. Moreover, each vertex v_i is associated with the following pre-computed data in $v_i.Aux$ (some of them are w.r.t. each possible radius $r \in [1, r_{max}]$):

- a keyword bit vector $v_i.BV_r$;
- a distance vector $v_i.Dist$;
- a support upper bound $v_i.ub\_sup_r$, and;
- a boundary influence upper bound $v_i.ub\_bound\_inf_r$.

Non-Leaf Nodes: Each non-leaf node $\mathcal{N}$ has multiple index entries, $\mathcal{N}_i$, each of which is associated with the following aggregates (w.r.t. each possible radius $r \in [1, r_{max}]$):

- a pointer to a child node $\mathcal{N}_i.ptr$;
- an aggregated keyword bit vector $\mathcal{N}_i.BV_r = \bigvee_{\forall v_l \in \mathcal{N}_i} v_l.BV_r$;
- the distance lower bound vector $\mathcal{N}_i.lb\_Dist$ (i.e., $N_i.lb\_Dist[j] = \min_{\forall v_l \in \mathcal{N}_i} v_l.Dist[j]$, for $1 \leq j \leq d$);
- the distance upper bound vector $\mathcal{N}_i.ub\_Dist$ (i.e., $N_i.ub\_Dist[j] = \max_{\forall v_l \in \mathcal{N}_i} v_l.Dist[j]$, for $1 \leq j \leq d$);
- the maximum support upper bound $\mathcal{N}_i.ub\_sup_r = \max_{\forall v_l \in \mathcal{N}_i} v_l.ub\_sup_r$, and;
- the maximum boundary influence upper bound $\mathcal{N}_i.ub\_bound\_inf_r = \max_{\forall v_l \in \mathcal{N}_i} v_l.ub\_bound\_inf_r$.

Index Construction: To construct the tree index $\mathcal{I}$, we will utilize cost models to first partition the graph into (disjoint) subgraphs of similar sizes to form initial leaf nodes, and then recursively group subgraphs (or nodes) into non-leaf nodes on a higher level, until one final root of the tree is obtained.

Cost Model for the Graph Partitioning: Specifically, we use METIS [23] for graph partitioning, guided by our proposed cost model. Our goal of designing a cost model for the graph partitioning is to reduce the number of cases that candidate communities are across subgraph partitions (or leaf nodes), and in turn achieve low query cost.

Assume that a graph partitioning strategy, $\mathcal{P}$, divides the graph into m subgraph partitions P_1, P_2, ..., and P_m. We can obtain the number, $Cross\_Par\_Size(\mathcal{P})$, of *cross-partition vertices for candidate communities* as follows.

$$\begin{aligned} & Cross\_Par\_Size(\mathcal{P}) \\ = & \sum_{j=1}^{m} \sum_{\forall v_i \in P_j} |V(r_{max}\text{-}hop(v_i, G) - P_j)| \end{aligned} \tag{5}$$

Since we would like to have the subgraph partitions of similar sizes, we also incorporate the maximum size difference of the resulting partitions in $\mathcal{P}$, and have the following target cost model, $CM(\mathcal{P})$.

$$CM(\mathcal{P}) = \arg\min_{\mathcal{P}} (Cross\_Par\_Size(\mathcal{P}) + (|\mathcal{P}_{max}| - |\mathcal{P}_{min}|)), \tag{6}$$

where $|\mathcal{P}_{max}|$ and $|\mathcal{P}_{min}|$ represent the numbers of users in the largest and smallest partitions in $\mathcal{P}$, respectively.

Intuitively, we would like to obtain a graph partitioning strategy $\mathcal{P}$ that minimizes our cost model $CM(\mathcal{P})$ (i.e., with low cross-partition search costs and of similar partition sizes, as given in Equation (6)).

Cost-Model-Guided Graph Partitioning for Obtaining Index Nodes: In Algorithm 3, we illustrate how to obtain a set, $\mathbb{S}_p$, of m graph partitions for creating index nodes, in light of our proposed cost model above. First, we randomly select m initial vertex pivots and form an initial set, $\mathbb{S}_{piv}$ (line 1). Then, we use $\mathbb{S}_{piv}$ to perform the graph clustering and obtain m partitions in $\mathbb{S}_p$ (line 2). We invoke calculate_cost($\mathbb{S}_\text{p}$) in Algorithm 4 to calculate the cost, $local\_cost$, of the partitioning $\mathbb{S}_p$ (i.e., via $CM(\mathbb{S}_p)$ in Equation (6) of our cost model; line 3).

Algorithm 3: Cost-Model-Guided Graph Partitioning for Index Nodes

```
Input: i) a social network $G$, ii) the number $m$ of center vertices for
       partitioning, and iii) the maximum number of iterations $iter_{max}$
Output: a set, $\mathbb{S}_p$, of $m$ graph partitions for creating index nodes
1  randomly select $m$ initial vertex pivots and form $\mathbb{S}_{piv}$
2  use $\mathbb{S}_{piv}$ clustering to form $m$ partitions $\mathbb{S}_p$
3  calculate the cost of the partitioning: $local\_cost$ = calculate_cost($\mathbb{S}_\text{p}$)
4  for $iter = 1$ to $iter\_max$ do
5      select a random pivot $piv \in \mathbb{S}_{piv}$
6      randomly select a new vertex $piv_{new}$ that satisfies the requirements of
       $\mathbb{S}_{piv}$
7      $\mathbb{S}_{piv}^{'} = \mathbb{S}_{piv} - \{piv\} + \{piv_{new}\}$
8      use $\mathbb{S}_{piv}^{'}$ clustering to form $m$ partitions $\mathbb{S}_p^{'}$
9      calculate the cost of new partitions: $cost_{new}$ = calculate_cost($\mathbb{S}_\text{p}^{'}$)
10     if $cost_{new} < local\_cost$ then
11         $\mathbb{S}_{piv} = \mathbb{S}_{piv}^{'}$
12         $\mathbb{S}_p = \mathbb{S}_p^{'}$
13         $local\_cost = cost_{new}$
14 return $\mathbb{S}_p$
```

Next, we perform $iter\_max$ iterations to find the best pivot set $\mathbb{S}_{piv}$ and graph partitioning $\mathbb{S}_p$ with low cost $local\_cost$ (lines 4-13). In each iteration, we randomly replace one of vertex pivots, piv, in $\mathbb{S}_{piv}$ with a new non-pivot vertex piv_{new}, forming a new pivot set, $\mathbb{S}'_{piv}$ (lines 5-7). This way, we can use $\mathbb{S}'_{piv}$ to perform the graph clustering and obtain m new partitions in $\mathbb{S}'_p$, so that we invoke the function calculate_cost($\mathbb{S}'_\text{p}$) to calculate a new cost, $cost_{new}$, of partitioning $\mathbb{S}'_p$ (lines 8-9). Correspondingly, if $cost_{new}$ is less than $local\_cost$, we accept the new partitioning strategy by updating $\mathbb{S}_{piv}$, $\mathbb{S}_p$, and $local\_cost$ with $\mathbb{S}'_{piv}$, $\mathbb{S}'_p$, and $cost_{new}$, respectively (lines 10-13). Finally, we return m subgraph partitions, $\mathbb{S}_p$, to create m index nodes, respectively (line 14).

Complexity Analysis: For the tree index $\mathcal{I}$, let γ denote the fanout of each non-leaf node $\mathcal{N}$. In $\mathcal{I}$, since the number of leaf nodes is equal to the number of vertices $|V(G)|$, the depth of tree index $\mathcal{I}$ is $\lceil \log_\gamma |V(G)| \rceil + 1$. The time complexity of cost-model-guided graph partitioning for index nodes is given by $O((|V(G)| \cdot m + |V(G)|) \cdot iter\_max)$. On the other hand, the time complexity of recursive tree index construction is $O((\gamma^{dep} - 1)/(\gamma - 1) \cdot Partitioning)$. Therefore, the time complexity of our tree index construction is given by $O((\gamma^{\lceil \log_\gamma |V(G)| \rceil + 1} - 1)/(\gamma - 1) \cdot |V(G)| \cdot (m + 1) \cdot iter\_max)$.

Algorithm 4: calculate_cost(·) Function

```
Input: a set, $\mathbb{S}_p$, of partitions over social network $G$
Output: a cost score, $CM(\mathcal{P})$, for the partitioning in $G$
$CM(\mathcal{P}) = 0$
for each $P \in \mathbb{S}_p$ do
    for each $v_i \in V(P)$ do
        count the number of vertices that cross the partition $P$'s range:
          $N\_cross = |V(r_{max}\text{-}hop(v_i, G) - P)|$
    $Cross\_Par\_Size(P) = \sum_{\forall v_i \in P} N\_cross$
    add the value of $Cross\_Par\_Size(P)$ to $CM(\mathcal{P})$
add $|\mathcal{P}_{max}| - |\mathcal{P}_{min}|$ to $CM(\mathcal{P})$
return $CM(\mathcal{P})$
```

6 ONLINE RICS COMPUTATION

In this section, we provide our online RICS computation algorithm in Algorithm 5, which traverses our constructed tree index $\mathcal{I}$ and retrieves the RICS community answer that has the highest influence on the target community Q, by seamlessly integrating our effective pruning strategies.

Section 6.1 presents effective pruning strategies on the node level of the tree index. Section 6.2 details our proposed online RICS query processing procedure.

6.1 Index Pruning

In this subsection, we present effective pruning methods on the index level, which are used to prune index nodes containing (a group of) community false alarms.

Keyword Pruning for Index Entries: The idea of our keyword pruning over index entries is as follows. If all the r-hop subgraphs under an index entry $\mathcal{N}_i$ do not contain any keywords in the query keyword set L_q, then the entire index entry $\mathcal{N}_i$ can be safely filtered out.

Below, we provide the *index keyword pruning* method that uses the aggregated keyword bit vector $\mathcal{N}_i.BV_r$ stored in $\mathcal{N}_i$.

Lemma 6.1. **(Index Keyword Pruning)** *Given an index entry $\mathcal{N}_i$ and a bit vector, $L_q.BV$, for the query keyword set L_q, the index entry $\mathcal{N}_i$ can be safely pruned, if it holds that $\mathcal{N}_i.BV_r \wedge L_q.BV = \mathbf{0}$.*

Proof. If $\mathcal{N}_i.BV_r \cap L_q.BV = \emptyset$ holds, which means that all communities in $\mathcal{N}_i$ do not contain any of the keywords in L_q. According to Definition 4, $\mathcal{N}_i$ cannot be a candidate seed community, so it can be safely pruned. □ □

Support Pruning for Index Entries: Next, we present the *index support pruning* method, which utilizes the maximum upper bound support $\mathcal{N}_i.ub\_sup_r$ of the index entry $\mathcal{N}_i$ and the given support k to rule out the entry with low support.

Lemma 6.2. **(Index Support Pruning)** *Given an index entry $\mathcal{N}_i$ and a support parameter k, the index entry $\mathcal{N}_i$ can be safely pruned, if it holds that $\mathcal{N}_i.ub\_sup_r < k$, where $\mathcal{N}_i.ub\_sup_r$ is the maximum support upper bound for all r-hop subgraphs under $\mathcal{N}_i$.*

Proof. $\mathcal{N}_i.ub\_sup_r$ is the maximum support upper bound in all r-hop subgraphs under index entry $\mathcal{N}_i$. If $\mathcal{N}_i.ub\_sup_r < k$ holds, then all support upper bounds of r-hop subgraphs under $\mathcal{N}_i$ are less than k. By the inequality transition, all the supports of r-hop subgraphs under entry $\mathcal{N}_i$ are thus less than k. Based on Definition 4, all r-hop subgraphs under $\mathcal{N}_i$ cannot be a candidate seed community. Therefore, index entry $\mathcal{N}_i$ can be safely pruned, which completes the proof of this lemma. □

□

6.2 The RICS Algorithm

In this subsection, we illustrate our online RICS processing algorithm by traversing the tree index $\mathcal{I}$ in Algorithm 5.

Initialization: First, our RICS algorithm obtains a query bit vector $L_q.BV$ by hashing all keywords from the query keyword set L_q (line 1). Then, according to the given query center vertex v_q, the algorithm determines the target community Q (line 2). After that, we initialize an empty community S to store the best seed community we have searched so far. Moreover, we maintain an initially empty set, M, which keeps a set of potential candidate communities for delayed refinement. We also set a variable, $max\_inf\_so\_far$, to 0, which indicates the highest influence score we have encountered so far for the early termination of the index traversal (line 3).

Index Traversal: To facilitate the index traversal, we maintain a *minimum heap* $\mathcal{H}$, which accepts heap entries in the form $(\mathcal{N}, key)$, where $\mathcal{N}$ is an index node, and key is the minimum lower bound of the distances from vertices under node $\mathcal{N}$ to query vertex v_q (line 4). To start the index traversal, we insert all entries in the root of index $\mathcal{I}$ into heap $\mathcal{H}$ (line 5). Then, we traverse the index by accessing entries from $\mathcal{H}$ in ascending order of distance lower bounds (intuitively, communities closer to v_q will have higher influences on Q; lines 6-31).

Each time, we pop out a heap entry $\mathcal{N}$ with the minimum key key from the heap (lines 6-7). Assume that $max\_inf\_ub(\mathcal{H})$ is the maximum influence score upper bound in the current heap $\mathcal{H}$. If $max\_inf\_so\_far \geq max\_inf\_ub(\mathcal{H})$ holds, then all the candidate communities in the heap $\mathcal{H}$ cannot have higher influences than those communities we have already obtained. Thus, in this case, we can terminate the index traversal (lines 8-9). Otherwise, we will continue to check entries in the index node $\mathcal{N}$.

Algorithm 5: Online RICS Processing

```
   Input: i) a social network $G$, ii) a set, $L_q$, of query keywords, iii) the
          maximum radius, $r$, of each community, iv) an integer parameter, $k$,
          of the truss for each seed community, v) an integer parameter, $N$, of
          the maximum user number for each seed community, vi) the query
          center vertex, $v_q$, and vii) the index $\mathcal{I}$
   Output: a seed community, $S$, with the highest influential score
   // initialization
 1 hash all keywords in the query keyword set $L_q$ into a query bit vector $L_q.BV$
 2 obtain the target community $Q = r\text{-}hop(v_q, G)$
 3 $S = \emptyset, M = \emptyset, max\_inf\_so\_far = 0$
   // the index traversal
 4 initialize a minimum heap $\mathcal{H}$ accepting index entries in the form $(\mathcal{N}, key)$
 5 insert all entries $\mathcal{N}$ in the root of index $\mathcal{I}$ into heap $\mathcal{H}$
 6 while $\mathcal{H}$ is not empty do
 7     $(\mathcal{N}, key) = \mathcal{H}.pop()$
 8     if $max\_inf\_so\_far \geq max\_inf\_ub(\mathcal{H})$ then
 9         terminal the loop
10     if $\mathcal{N}$ is a leaf node then
11         for each vertex $v_i \in \mathcal{N}$ do
12             obtain the candidate community $C = \mathcal{N}.r\text{-}hop(v_i, G)$
13             if $C$ cannot be pruned by Lemma 4.1, 4.2, or 4.3 then
14                 if $C$ is closer to $Q$ than $S$ and is larger than $S$ then
15                     compute the influence score
                       $inf\_score_{C,Q}$ = calculate_influence$(C, Q)$
16                     if $inf\_score_{C,Q} > max\_inf\_so\_far$ then
17                         if $|V(C)| \leq N$ then
18                             $max\_inf\_so\_far = inf\_score_{C,Q}$
19                             $S = C$
20                         else
21                             update $C$ to be the $k$-truss subgraph of $C$
                                 of size $N$ with maximal influence score
                                 $inf\_score_{C,Q}$
22                             if $inf\_score_{C,Q} > max\_inf\_so\_far$
                                 then
23                                 $max\_inf\_so\_far = inf\_score_{C,Q}$
24                                 $S = C$
25                 else
26                     if $v_i.ub\_inf\_score_r > max\_inf\_so\_far$ then
27                         add $C$ to $M$
28     else
           // $\mathcal{N}$ is a non-leaf node
29         for each entry $\mathcal{N}_i \in \mathcal{N}$ do
30             if $\mathcal{N}_i$ cannot be pruned by Lemma 6.1 or 6.2 then
31                 insert $(\mathcal{N}_i, key)$ into heap $\mathcal{H}$
   // refinement of candidate communities
32 update $M$ by sorting on influence score upper bounds
33 for each candidate community $C \in M$ do
34     if $max\_inf\_so\_far > C.ub\_inf\_score_r$ then
35         terminal the loop
36     select no more than $N$ vertices to form a community $C'$ from $C$
37     compute the influence score
       $inf\_score_{C',Q}$ = calculate_influence(C', Q)
38     if $inf\_score_{C',Q} > max\_inf\_so\_far$ then
39         $max\_inf\_so\_far = inf\_score_{C',Q}$
40         $S = C'$
41 return $S$
```

When $\mathcal{N}$ is a leaf node, for each vertex $v_i \in \mathcal{N}$, we obtain its candidate community $C = r\text{-}hop(v_i, G)$ centered at v_i. After that, for candidate community C, we apply the *Keyword Pruning* (Lemma 4.1), *Support Pruning* (Lemma 4.2), and *Influence Score Pruning* (Lemma 4.3) (lines 10-13). If C cannot be ruled out by these three pruning methods, then we will prefer candidate communities that are closer to the target community Q than the current optimal seed community S and larger than the size of S (this is a heuristic optimization method, and intuitively, such candidate communities have more influence on Q), after that, we calculate the exact influence, $inf\_score_{C,Q}$, from C to target community Q, by invoking the function calculate_influence(C, Q) (lines 14-15). If the influence $inf\_score_{C,Q}$ is higher than the highest influence score, $max\_inf\_so\_far$, and the seed community C is of small scale (i.e., $\leq N$), we will update the current answer S and its influence score $max\_inf\_so\_far$ (lines 16-19). If the seed community C is of large scale (i.e., $> N$), we will update C to be the k-truss subgraph of C of size N with the largest influence score $inf\_score_{C,Q}$ of all k-truss subgraphs (lines 20-21). Then, like the small scale, if the influence $inf\_score_{C,Q}$ is higher than the highest influence score, $max\_inf\_so\_far$, we will update the current answer S and the highest influence score $max\_inf\_so\_far$ (lines 22-24). On the other hand, although C is not better than S in terms of position and size, it also has the potential to have the highest influence on Q. If its influence score upper bound (for any subgraphs of size N) is greater than the highest influence, we will add C to the candidate set M for later refinement (lines 25-27).

When $\mathcal{N}$ is a non-leaf node, we will consider each child node $\mathcal{N}_i \in \mathcal{N}$ (lines 28-29). If entry $\mathcal{N}_i$ cannot be pruned by *Index Keyword Pruning* (Lemma 6.1) and *Index Support Pruning* (Lemma 6.2), we insert the entry $(\mathcal{N}_i, key)$ into heap $\mathcal{H}$ for further investigation (lines 30-31).

When either the heap $\mathcal{H}$ is empty (line 6) or the remaining index entries in $\mathcal{H}$ cannot contain candidate communities (line 8), we will terminate the index traversal.

Refinement of Candidate Communities: After the index traversal, we update M by sorting candidate communities in descending order of influence score upper bounds (line 32). Then, for each candidate community C in the list M, if it holds that $max\_inf\_so\_far > C.ub\_inf\_score_r$, then we can stop checking the remaining candidates in M (as all candidates in M have influence upper bounds less than highest influence so far; lines 34-35). Next, we compute a seed community C' of size no more than N from C (i.e., $C' \subset C$) with the highest influence, $inf\_score_{C',Q}$, on target community Q (lines 36-37). If it holds that $inf\_score_{C',Q} > max\_inf\_so\_far$, then we need to update the RICS answer S with C' (replacing $max\_inf\_so\_far$ with $inf\_score_{C',Q}$ as well; lines 38-40). Finally, after the refinement of M, we return the seed community S as our RICS answer (line 41).

Discussions on the Computation of calculate_influence(C, Q)**:** To exactly calculate the community-to-community influence score (via Equation (3) and (4)), we need to obtain the influence of each user in the seed community C on the target community Q, the whole process is similar to the single-source shortest path algorithm. For each point v_c in C, we first visit its 1-hop neighbors v_b and the influence score $inf\_score_{vc,vb} = P_{vc,vb}$. Then, each time, we extend 1-hop neighbors v_{new} forward and compute the current influence score $inf\_score_{vc,v_{new}} = \max_{\forall v_i \in v_{new}}(inf\_score_{v_{new},v_i} \cdot P_{v_{new},v_i})$, of v_c, until we get the maximum influence score on all node of Q.

Discussions on the Online Computation of Influence Upper Bound $v_i.ub\_inf\_score_r$: Since we get the upper bound of boundary influence score, $v_i.ub\_bound\_inf_r$, of collapse_calculate data for a subgraph $r\text{-}hop(v_i, G)$, and for a target community, Q, with query center vertex, v_q, we can get the distance lower bound $lb\_dist(v_i, v_q) = \min([|v_i.Dist[j] - v_q.Dist[j]|]$, for $1 \leq j \leq d)$ between v_i and v_q by *triangle inequality* [24]. Then, we can get the upper bound of influence score, $v_i.ub\_inf\_score_r = v_i.ub\_bound\_inf_r \cdot |V(Q)| \cdot \max(P)^{lb\_dist(v_i,v_q)-2\cdot r}$, where $\max(P)$ denotes the maximum neighbor activation probability in G.

Complexity Analysis: Let $\overline{n_r}$ be the average number of users in the target community Q. The cost of obtaining $L_q.BV$ and Q takes $O(\overline{n_r})$. Let PP_j be the pruning power (i.e., the percentage of node entries that can be pruned) on the j-th level of the tree index $\mathcal{I}$, where $0 \leq j \leq h$ and h is the height of the tree. Denote f as the average fanout of nodes in index $\mathcal{I}$. For the index traversal, the number of visited nodes is given by $O(\sum_{j=1}^{h} f_{h-j+1} \cdot (1 - PP_j))$. We label a subgraph $r\text{-}hop(v_i, G)$ as g. Each time, the function of calculate_influence need $O((V|g|+E|g|)\cdot\overline{n_r})$. Let $\overline{n_d}$ be the average number of iterations updated due to the closest distance. Then, the updating S and $max\_inf\_so\_far$ takes $O((V|g|+E|G|)\cdot\overline{n_r}\cdot\overline{n_d})$. And, updating M takes $O(1)$. For the refinement process, let $\overline{n_m}$ be the average number of calculate influence in M, and the updating S and $max\_inf\_so\_far$ take $O(1)$. Therefore, the total time complexity of Algorithm 5 is given by $O(\sum_{j=1}^{h} f_{h-j+1} \cdot (1 - PP_j) + ((V|g| + E|g|) \cdot (\overline{n_d} + \overline{n_m}) + 1) \cdot \overline{n_r})$.

7 ONLINE COMPUTATION FOR R^{2}ICS

A Framework for the R^{2}ICS Algorithm. Our R^{2}ICS algorithm has two steps. First, we will invoke the RICS index traversal algorithm without pruning to obtain as many as N high influence vertices as possible in the initialization phase, and the other candidate vertices will add to the refinement set M to refine candidate vertices (lines 1-3). Then, we do a descending sort of M by influence upper bound and replace S cyclically by the minimum of the obtained influence score values of S until we obtain the maximum influence community S that satisfies N (lines 4-21).

Effective Pruning Strategy w.r.t. Keyword and Influence Upper Bound. To accurately find the highest influential community in the whole graph, in the worst case, it is necessary to traverse all the vertices to find the community that meets the requirement, but it is too inefficient to be applied in real-world application.

To reduce the search space, we propose an effective *R^{2}ICS refine influence score pruning* method, which can avoid searching all vertices in G. And we give the following pruning lemma:

Lemma 7.1. ***(R^{2}ICS Refine Influence Score Pruning)*** *Let an influence threshold θ be the minimum influence we have obtained from the maximum set of influence vertices to the target community Q. Any vertex v can be safely pruned, if it holds that $ub\_inf\_score_{v,Q} < \theta$, where $ub\_inf\_score_{v,Q}$ is an upper bound of the influence score $inf\_score_{v,Q}$.*

Proof. Since $ub\_inf\_score_{v,Q}$ is an upper bound of the influence score $inf\_score_{v,Q}$, we have $inf\_score_{v,Q} \geq ub\_inf\_score_{v,Q}$. If $ub\_inf\_score_{v,Q} < \theta$ holds, means $inf\_score_{v,Q} < \theta$, which indicates that the vertex v has a lower influence on Q, compared with some vertices we have obtained so far (i.e., with influence θ), and v cannot be our R^{2}ICS answer. Therefore, we can safely prune candidate vertex v, which completes the proof. □ □

The Online R^{2}ICS Algorithm. Algorithm 6 shows the pseudocode to handle the online R^{2}ICS query over a given social network G. The whole algorithm is divided into initialization and refinement computation phases.

Initialization: Specifically, after hashing all keywords and obtaining the target community, Q (lines 1-2), we invoke the RICS index traversal algorithm without pruning (i.e., Algorithm 5) to obtain a set of vertices with maximum influence on Q that is no more than N, S, the corresponding set of influence scores, S_inf, and a set of candidate vertices, M, prepare for the subsequent refinement computation phases (line 3).

Refinement Computation: To refine candidate vertices in M, we first update M by sorting on influence score upper bound for each vertex in M (line 4). And, we will maintain a minimum value $min\_inf\_so\_far$ for the currently obtained result and initialize to $\min(S\_inf)$ (line 5). Then, each time, we will check whether or not the vertices set length $len(S)$ is gather than N, considering the following two cases (lines 6-20):

Algorithm 6: Online R^{2}ICS Processing

```
Input: i) a social network $G$, ii) a set, $L_q$, of query keywords, iii) the
       maximum radius, $r$, of target community, iv) an integer parameter, $N$,
       of the maximum user number for the highest influence community, v)
       the query center vertex, $v_q$, and vi) the index $\mathcal{I}$
Output: a community, $S$, with the highest influential score
// initialization
1  hash all keywords in the query keyword set $L_q$ into a query bit vector $L_q.BV$
2  obtain the target community $Q = r\text{-}hop(v_q, G)$
3  Invoke the RICS index traversal algorithm without pruning to obtain a set, $S$,
     of vertices with maximum influence on $Q$, no more than $N$, the
     corresponding set of influence scores, $S\_inf$, and a set of candidate vertices,
     $M$.
// refinement computation
4  update $M$ by sorting on influence score upper bounds
5  maintain a minimum value for the currently obtained result and initialize
     $min\_inf\_so\_far = \min(S\_inf)$
6  for each candidate vertex $v_i \in M$ do
7      if $len(S) \geq N$ then
8          if $v_i$ cannot be pruned by Lemma 7.1 then
9              compute the influence score
                 $inf\_score_{v_i,Q} = \text{calculate\_influence}(v_i, Q)$
10             if $inf\_score_{v_i,Q} > min\_inf\_so\_far$ then
11                 update the vertex of $min\_inf\_so\_far$ in $S$ as $v_i$
12                 update the influence score of $min\_inf\_so\_far$ in $s\_inf$
                     as $inf\_score_{v_i,Q}$
13                 update $min\_inf\_so\_far = \min(S\_inf)$
14         else
15             terminal the loop
16     else
17         compute the influence score
             $inf\_score_{v_i,Q} = \text{calculate\_influence}(v_i, Q)$
18         add $v_i$ to $S$
19         add $inf\_score_{v_i,Q}$ to $S\_inf$
20         update $min\_inf\_so\_far = \min(S\_inf)$
21 return $S$
```

Case 1: If $len(S) \geq N$ holds, we will execute Lemma 7.1 to determine whether to terminate the loop early (lines 7-8). If v_i is pruned off, then none of the higher influence vertices will exist, and we can safely terminate (lines 14-15). If there is no pruning off, then we will calculate the exact influence score $inf\_score_{v_i,Q}$ of v_i on Q. If $inf\_score_{v_i,Q}$ is greater than $min\_inf\_so\_far$, we will replace v_i with the vertex corresponding to the $min\_inf\_so\_far$ in S, and similarly S_inf, and then update $min\_inf\_so\_far$ to be the minimum value of the current S_inf (lines 9-13).

Case 2: If $len(S) < N$ holds, it indicates that we still have enough spaces for other vertices. Then, we will compute the influence score $inf\_score_{v_i,Q}$ of v_i on Q, and add v_i, $inf\_score_{v_i,Q}$ to S and S_inf, respectively, and then update the $min\_inf\_so\_far$ (lines 16-20).

After refinement, we obtain the most influential up to N candidate vertices and return S as the R^{2}ICS answer.

Complexity Analysis: Since we invoked the index traversal of the Algorithm 5, the initialization of our R^{2}ICS takes $O(\sum_{j=1}^{h} f_{h-j+1} + (V|g| + E|g| + 1) \cdot \overline{n_r} \cdot N)$. For refinement, let GPP be the R^{2}ICS refine influence pruning power (i.e., the percentage of vertices that can be pruned). And, updating S, S_inf and $min\_inf\_so\_far$ takes $O(1)$. Therefore, the total time complexity of Algorithm 6 is given by $O(\sum_{j=1}^{h} f_{h-j+1} + ((V|g| + E|g|) \cdot (N + |V(G)| \cdot GPP) + 1) \cdot \overline{n_r})$

Table 2: Statistics of the tested real-world graph data sets.

Social Networks	$\lvert V(G)\rvert$	$\lvert E(G)\rvert$
Facebook[25]	4,039	88,234
Amazon[26]	334,863	925,872
DBLP[27]	317,080	1,049,866

Table 3: Parameter settings.

Parameters	Values
support, k, of truss structure	3, **4**, 5
radius r	1, **2**, 3
size, $\lvert L_q\rvert$, of query keywords set	2, 3, **5**, 8, 10
size, $\lvert v_i.L\rvert$, of keywords per vertex	1, 2, **3**, 4, 5
keyword domain size $\lvert\Sigma\rvert$	10, **20**, 50, 80
the number, d, of pivots	3, **5**, 8, 10
the maximum size, N, of seed community	5, **10**, 15, 20
the size, $\lvert V(G)\rvert$, of data graph G	10K, 25K, **50K**, 100K, 250K

8 EXPERIMENTAL EVALUATION

8.1 Experimental Settings

We evaluate the performance of the online RICS algorithm (i.e., Algorithm 5) on both real and synthetic graph data sets.

Real-World Graph Data Sets: We use three real-world graphs, Facebook [25], Amazon [26], and DBLP [27], whose statistics are depicted in Table 2. Facebook is a social network, where two users are connected if they are friends. Amazon is an Also Bought network, where two products are connected if they are purchased together. DBLP is a co-authorship network, where two authors are connected if they publish at least one paper together.

Synthetic Graph Data Sets: We construct synthetic social networks by generating *small-world graphs* G [28]. Specifically, we first create a ring of size $|V(G)|$, and then connect m nearest neighbor nodes for each vertex u. Next, for each generated edge $e_{u,v}$, we add a new edge $e_{u,w}$ with probability μ that connects u to a random vertex w. Here, we take $m = 5$ and $\mu = 0.251$. For each vertex, we randomly generate a keyword set $v_i.L$ from the keyword domain Σ, following $Uniform$, $Gaussian$, and $Zipf$ distributions, to obtain three synthetic graphs, denoted as Uni, Gau, and $Zipf$, respectively. Next, for each edge $e_{u,v}$ in the generated graphs, we produce a random value within an interval $[0.5, 0.6)$ as the edge activation probability $P_{u,v}$.

Competitors: To our best knowledge, no prior works studied the RICS problem and its variant R^{2}ICS problem by considering the influence of a connected community on a user-specified target community (instead of the entire graph). Therefore, we compare our RICS approach with a straightforward method, called *baseline*. The *baseline* method first determines the target community Q based on the given query vertex and then performs *Breadth First Search* (BFS) from Q in the social network G. For each vertex we encounter (during the BFS traversal), *baseline* obtains its r-hop subgraph and checks whether this subgraph satisfies the structure and keyword constraints. Next, we obtain candidate communities S from the r-hop subgraph and calculate their influence scores $inf\_score_{S,Q}$. If a candidate community S has an influence score greater than the best score we have seen so far, we will let S be the best-so-far RICS answer. Finally, after all vertices have been traversed, *baseline* returns the candidate community S we have obtained with the maximum influence on the target community Q. Note that,

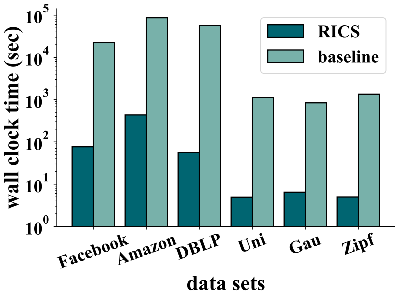


Figure 3: The RICS performance on real/synthetic graphs.

since the time cost of the *baseline* method is extremely high, we evaluate this method by sampling 0.1% vertices from the data graph G without replacement. Therefore, the total time can be estimated by $\bar{t} \cdot |V(G)|$, where $\bar{t}$ denotes the average time of each sample.

For R^{2}ICS, we compare our approach (R^{2}ICS, Algorithm 6) with R^{2}ICS_WoP and *Optimal* methods. R^{2}ICS_WoP is the R^{2}ICS without pruning strategy in Section 7, where *Optimal* computes the influence score of each vertex on the target community in the original graph and selects the largest combination of influence scores as a result.

Measure: To evaluate the efficiency of our RICS approach, we randomly select 50 query nodes from each dataset, and finally, we report average *wall clock time*, which is the time cost of online retrieving RICS query results via the index (Algorithm 5).

Parameter Settings: Table 3 depicts parameter settings, where default values are in bold. Each time, we vary the value of one parameter while setting other parameters to their default values. We ran all the experiments on the PC with Intel(R) Core(TM) i9-10900K CPU 3.70GHz and 32 GB memory. All algorithms were implemented in Python and executed with Python 3.8 interpreter.

8.2 Performance Evaluation

The RICS Performance on Real/Synthetic Graphs: Figure 3 illustrates the performance of our RICS approach on both real and synthetic graphs, compared with the *baseline* method, where all parameters are set by their default values in Table 3, except for the dense *Facebook* dataset with the maximum seed community size $N = 700$. Experimental results show that the *wall clock time* of our RICS approach outperforms *baseline* by almost four orders of magnitude, which confirms the effectiveness of our proposed pruning strategies and indexing mechanisms and the efficiency of our RICS approach.

To evaluate the robustness of our RICS approach, in subsequent experiments, we will test the effect of each parameter in Table 3 on the query performance over synthetic graphs.

Effect of Truss Support Parameter k: Figure 4(a) shows the RICS query performance for different k values, where k = 3, 4, and 5, and the rest of parameters are set to default values. From this figure, we can find that for larger k values, the query time cost decreases over all three synthetic graphs. This is because larger k leads to fewer candidate communities satisfying the k-truss constraints, which in turn incurs lower wall clock time.

Effect of Radius r: Figure 4(b) illustrates the wall clock time of our RICS approach, by varying r from 1 to 3, where other parameters are set to their default values. When the radius r increases, the numbers of vertices included in the target and seed communities also increase, leading to higher filtering and refinement costs. Nevertheless, the wall clock time remains low (i.e., 3.54 ~ 17.44 *sec*) for all the synthetic graphs.

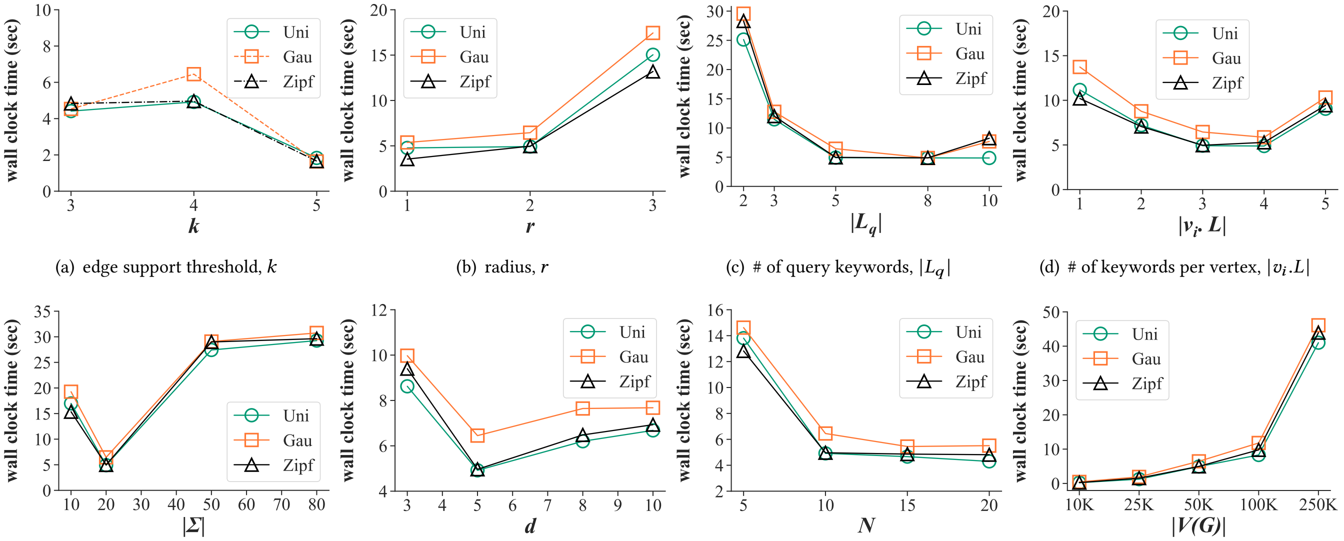
Effect of the Size, $|L_q|$, of the Query Keyword Set: Figure 4(c) presents the RICS query performance, where $|L_q|$ = 2, 3, 5, 8, and 10, and other parameters are by default. Intuitively, as $|L_q|$ increases, more candidate seed communities satisfy the keyword requirements. Thus, we will have a higher threshold of the influence score, which results in higher pruning power and, in turn, lower time cost, as confirmed by the figure. However, more candidate seed communities require more refinement costs, and wall clock times increase. In summary, the wall clock time remains low for different $|L_q|$ values (i.e., 4.86 ~ 29.56 *sec*).

Effect of the Size, $|v_i.L|$, of Keywords per vertex: Figure 4(d) reports the efficiency of our RICS approach, by varying $|v_i.L|$ from 1 to 5, where default values are used for other parameters. With the increase of $|v_i.L|$, more vertices are likely to be included in candidate seed communities, which leads to a higher influence threshold and higher pruning power (or lower query cost). Meanwhile, larger $|v_i.L|$ will incurs higher filtering/refinement costs. Therefore, the two factors mentioned above show that the wall clock time first decreases and then increases for larger $|v_i.L|$. The wall clock times with different $|v_i.L|$ values are 4.88 ~ 13.73 *sec*.

Effect of Keyword Domain Size $|\Sigma|$: Figure 4(e) illustrates the RICS query performance with different keyword domain sizes $|\Sigma|$ = 10, 20, 50, and 80, where other parameters are set to default values. From this figure, we can find that, since larger Σ will improve the pruning power of keyword pruning, the community computational cost decreases. On the other hand, fewer candidate communities also lead to lower impact thresholds and lower pruning power. Thus, for all three synthetic graphs, the wall clock time decreases and then increases as Σ increases. Nevertheless, the wall clock times remain low (i.e., 4.92 ~ 30.75 *sec*).

Effect of the Number, d, of Pivots: Figure 4(f) shows the RICS query performance for various numbers of pivots, where d = 3, 4, 5, 6, and 8, and default values are used for other parameters. When d increases, the distance lower bounds from candidate communities S to target community Q are tighter, which incurs better searching order of candidate communities and achieves higher influence threshold earlier (or lower query costs). However, more pivots will also lead to higher computation costs for distances with lower bounds. Therefore, in the figure, for larger d values, the wall clock time first decreases and then increases. Nonetheless, the wall clock times remain low (i.e., 4.92 ~ 9.97 *sec*).

Effect of the Maximum Size, N, of Seed Communities: Figure 4(g) evaluates the performance of our RICS approach, where the maximum size, N, of seed communities varies from 5 to 20, and other parameters are by default. The smaller N is, while we have fewer candidate communities, the computational cost of the k-truss subgraph with maximum influence performed to obtain these candidate communities is greatly increased. Therefore, in the figure, when N increases, the wall clock time decreases for all the three synthetic graphs. Nevertheless, the time costs remain low (i.e., 4.30 ~ 14.62 *sec*) for different N values.



(a) edge support threshold, k

(b) radius, r

(c) # of query keywords, $|L_q|$

(d) # of keywords per vertex, $|v_i.L|$

(e) keyword domain size, $|\Sigma|$

(f) # of pivots, d

(g) maximum seed community size, N

(h) graph size, $|V(G)|$

Figure 4: The robustness evaluation of the RICS query performance.

Effect of the Size, $|V(G)|$, of the Data Graph G: Figure 4(h) tests the scalability of our RICS approach, where graph size $|V(G)| = 10K$, $25K$, $50K$, $100K$, and $250K$, and the rest of parameters are set by their default values. From the figure, we can see that, with the increase of the graph size $|V(G)|$, the number of candidate seed communities also increases, which leads to higher pruning/refinement costs and more wall clock times. Nonetheless, even when $|V(G)| = 250K$ (i.e., $250K$ vertices in graph G), the time costs are less than 46.10 *sec* for all the three synthetic graphs, which confirms the efficiency and scalability of our proposed RICS approach on large-scale social networks.

The R^{2}ICS Performance on Real/Synthetic Graphs: Figure 5 illustrates the performance of our R^{2}ICS approach on both real and synthetic graphs, compared with the R^{2}ICS_WoP and *Optimal* methods, where all parameters are set by their default values in Table 3. From the figure, since R^{2}ICS uses Lemma 7.1 as an influence upper bound pruning strategy, it is unnecessary to specifically compute candidate vertices with a very small influence upper bound on the query target community. And, we can see the *wall clock time* of R^{2}ICS approach outperforms R^{2}ICS_WoP by about one order of magnitude and outperforms *Optimal* by about two orders of magnitude. Moreover, every vertex is fully considered in our refinement process, and the accuracy of our method is 100% as in *Optimal*. These results confirm our overall method's effectiveness and our R^{2}ICS's efficiency on real and synthetic graphs.

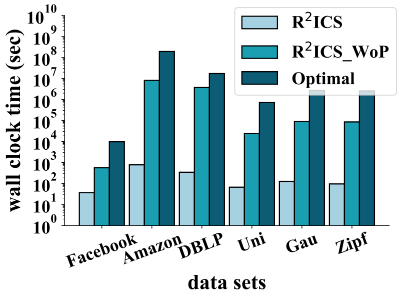
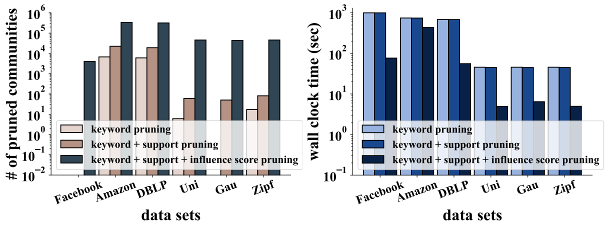


Figure 5: The R^{2}ICS performance on real/synthetic graphs.



(a) pruning power

(b) time cost

Figure 6: The ablation study of the RICS performance.

8.3 Ablation Study

To evaluate the effectiveness of our proposed pruning strategies, we conduct an ablation study over real/synthetic graphs, where all parameters are set to their default values. As shown in Figure 6(a) and 6(b), we tested different combinations by adding one more pruning strategy each time: (1) *keyword pruning* only, (2) *keyword + support pruning*, and (3) *keyword + support + influence score pruning*. Figure 6(a) shows the number of pruned candidate communities for different pruning combinations, and Figure 6(b) shows the query time cost with different pruning combinations. From these figures, we can find that as more pruning strategies are used, the number of pruned communities increases by 1-3 orders of magnitude, and the wall clock time also decreases by 1 order of magnitude accordingly. Especially, the third *influence score pruning* strategy can significantly prune more candidate communities and reduce the query cost. On the other hand, due to the high density of the

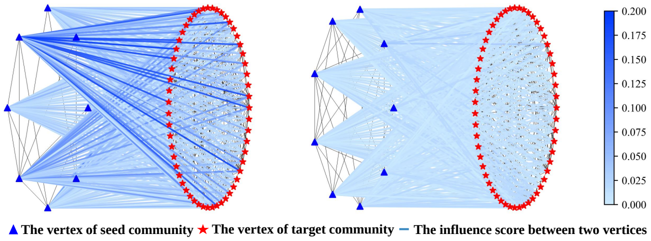

0.200
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
▲ The vertex of seed community ★ The vertex of target community — The influence score between two vertices

Figure 7: A case study of RICS with different community structures over DBLP dataset.

Facebook dataset, the number of pruned candidate communities for keyword and support pruning is zero. For *Gaussian*, since most of the vertices have the same keywords around the center vertex, the pruning power of the keyword pruning is zero.

8.4 Case Study

To evaluate the usefulness of our RICS results, we conduct a case study to compare the influences of the seed community obtained by our RICS approach with that by k-core [29] over *DBLP*. Figure 7 shows the visualization of influence propagation between the seed community and the target community, where blue triangles are the vertices of the seed community, red stars are the vertices of the target community, and shades of edge color reflect influence scores. With the same target community, the left part of Figure 7 is the result of our RICS approach (4-truss), and the right part is the result of the k-core method (4-core). From this figure, we can find that although the 4-core community has more vertices, our RICS seed community has an influence score of 15.74, significantly greater than the 4.72 of the 4-core community. This confirms the usefulness of our RICS problem to obtain seed communities with high influences for real-world applications such as online advertising/marketing.

9 RELATED WORK

In this section, we briefly discuss research closely related to our work, specifically community search, community detection, and influence maximization.

Community Search (CS): The *community search* (CS) over social networks usually search for connected subgraphs containing a specific query vertex or a set of query vertices [30–32]. Some works [33–37] typically adopt cohesive subgraph models to measure the cohesiveness of subgraphs as a way to obtain a community from a query vertex q, such as k-core [34, 35], k-truss [21, 36], k-clique [37, 38] and k-edge connectivity components [39, 40]. In [33, 34], the minimum degree was used to measure the cohesion of the k-core communities. In contrast, our RICS problem is more challenging in searching for densely structured communities and ensuring the high influence of communities with the constraint of query keywords. On the other hand, most studies on community search are searching from a certain user (forward search [32]), while our work focuses on reverse search starting from a certain community, which is more broadly considered and closer to real life.

Community Detection (CD): The *community detection* (CD) aims to detect all communities in a given social network. The foundation of many detection algorithms lies in graph partitioning [41, 42] and clustering [43–45]. The Kernighan-Lin algorithm [41] is one of the earliest techniques used for graph partitioning, which divides the nodes of the graph into smaller components with specific attributes while minimizing the number of cut edges. Newman's maximum likelihood algorithm [42], on the other hand, reduces the community detection problem to searching among a set of candidate solutions, each of which is a solution to the minimum cut graph partitioning. For the clustering method, Blondel et al. [45] presents a hierarchical clustering approach to address the CD problem, while Clauset et al. [44] notice modularity optimization and propose a greedy modularity optimization strategy to solve the CD problem. Different from CD, our RICS problem requires not just detecting communities but also finding the seed community that has the most influence on the target community.

Influence Maximization (IM): The *influence maximization* (IM) problem has been studied for a long time, which identifies a set of users as seed vertices with the maximum impact on other users within a given social network. Two influence propagation models proposed by Kempe et al. [46], the *Independent Cascade* (IC) model and the *Linear Threshold* (LT) model, have been widely used as influence propagation models for addressing influence maximization problems [20, 47–49]. Chen et al. [20] introduces the *DegreeDiscount* heuristic algorithm for LT, presenting a scalable influence maximization algorithm. [47] proposed the PMIA heuristic algorithm for the IC model. However, such IM problems typically ignore the constraints among seed vertices, whereas our RICS problem pays attention to identifying seed communities that can influence the given target user group.

10 CONCLUSION

This paper proposed a novel RICS problem, which returns a seed community with the maximum influence on a user-specified target community. Unlike existing works, the RICS problem considers the influence of seed community on a specific user group/community rather than arbitrary users in social networks. To solve the RICS problem, we designed effective pruning strategies to filter out false alarms of candidate seed communities, and constructed an index to facilitate our proposed efficient RICS query processing algorithm. We also formulated and tackled a variant of RICS (i.e., R^{2}ICS) by proposing an online query algorithm with effective refinement influence score pruning. Extensive experiments on real/synthetic social networks validated the efficiency and effectiveness of our RICS and R^{2}ICS approaches.

REFERENCES

[1] S. Tu and S. Neumann, “A viral marketing-based model for opinion dynamics in online social networks,” in *Proceedings of the Web Conference*, 2022, pp. 1570–1578.

[2] P. Ebrahimi, M. Basirat, A. Yousefi, M. Nekmahmud, A. Gholampour, and M. Fekete-Farkas, “Social networks marketing and consumer purchase behavior: the combination of sem and unsupervised machine learning approaches,” *Big Data and Cognitive Computing*, vol. 6, no. 2, p. 35, 2022.

[3] N. Rai and X. Lian, “Top-k community similarity search over large-scale road networks,” *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 35, no. 10, pp. 10 710–10 721, 2023.

[4] R. Molaei, K. R. Fard, and A. Bouyer, “Time and cost-effective online advertising in social internet of things using influence maximization problem,” *Wirel. Networks*, vol. 30, no. 2, pp. 695–710, 2024. [Online]. Available: https://doi.org/10.1007/s11276-023-03496-1

[5] S. Kumar, A. Mallik, A. Khetarpal, and B. Panda, “Influence maximization in social networks using graph embedding and graph neural network,” *Information Sciences*, vol. 607, pp. 1617–1636, 2022.

[6] N. Subramani, S. Veerappampalayam Easwaramoorthy, P. Mohan, M. Subramanian, and V. Sambath, “A gradient boosted decision tree-based influencer prediction in social network analysis,” *Big Data and Cognitive Computing*, vol. 7, no. 1, p. 6, 2023.

[7] R.-H. Li, L. Qin, J. X. Yu, and R. Mao, “Influential community search in large networks,” *Proceedings of the VLDB Endowment*, vol. 8, no. 5, pp. 509–520, 2015.

[8] R. Yan, D. Li, W. Wu, D. Du, and Y. Wang, “Minimizing influence of rumors by blockers on social networks: Algorithms and analysis,” *IEEE Trans. Netw. Sci. Eng.*, vol. 7, no. 3, pp. 1067–1078, 2020. [Online]. Available: https://doi.org/10.1109/TNSE.2019.2903272

[9] Y. Zhou, Y. Fang, W. Luo, and Y. Ye, “Influential community search over large heterogeneous information networks,” *Proceedings of the VLDB Endowment*, vol. 16, no. 8, pp. 2047–2060, 2023.

[10] M. S. Islam, M. E. Ali, Y.-B. Kang, T. Sellis, F. M. Choudhury, and S. Roy, “Keyword aware influential community search in large attributed graphs,” *Information Systems*, vol. 104, p. 101914, 2022.

[11] Y. Wu, J. Zhao, R. Sun, C. Chen, and X. Wang, “Efficient personalized influential community search in large networks,” *Data Science and Engineering*, vol. 6, no. 3, pp. 310–322, 2021.

[12] A. Al-Baghdadi and X. Lian, “Topic-based community search over spatial-social networks,” *Proceedings of the VLDB Endowment*, vol. 13, no. 12, pp. 2104–2117, 2020.

[13] D. Li, L. Zeng, R. Hu, X. Liang, and Y. Zang, “Itc: Influential-truss community search,” in *2022 International Joint Conference on Neural Networks (IJCNN)*, 2022, pp. 01–08.

[14] J. Xu, X. Fu, Y. Wu, M. Luo, M. Xu, and N. Zheng, “Personalized top-n influential community search over large social networks,” *World Wide Web (WWW)*, vol. 23, pp. 2153–2184, 2020.

[15] Q. Fang, J. Sang, C. Xu, and Y. Rui, “Topic-sensitive influencer mining in interest-based social media networks via hypergraph learning,” *IEEE Transactions on Multimedia*, vol. 16, no. 3, pp. 796–812, 2014.

[16] S. M. Firestone, M. P. Ward, R. M. Christley, and N. K. Dhand, “The importance of location in contact networks: Describing early epidemic spread using spatial social network analysis,” *Preventive Veterinary Medicine*, vol. 102, no. 3, pp. 185–195, 2011.

[17] H. Min, J. Cao, T. Yuan, and B. Liu, “Topic based time-sensitive influence maximization in online social networks,” *World Wide Web (WWW)*, vol. 23, pp. 1831–1859, 2020.

[18] K. Ali, C.-Y. Wang, and Y.-S. Chen, “Leveraging transfer learning in reinforcement learning to tackle competitive influence maximization,” *Knowledge and Information Systems*, vol. 64, no. 8, pp. 2059–2090, 2022.

[19] W. Chen, Y. Wang, and S. Yang, “Efficient influence maximization in social networks,” in *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, 2009, pp. 199–208.

[20] W. Chen, C. Wang, and Y. Wang, “Scalable influence maximization for prevalent viral marketing in large-scale social networks,” in *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2010, pp. 1029–1038.

[21] J. Cohen, “Trusses: Cohesive subgraphs for social network analysis,” *National Security Agency Technical Report*, vol. 16, no. 3.1, 2008.

[22] X. Huang and L. V. Lakshmanan, “Attribute-driven community search,” *Proceedings of the VLDB Endowment*, vol. 10, no. 9, pp. 949–960, 2017.

[23] G. Karypis and V. Kumar, “A fast and high quality multilevel scheme for partitioning irregular graphs,” *SIAM Journal on Scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[24] D. A. Plaisted, “Heuristic matching for graphs satisfying the triangle inequality,” *J. Algorithms*, vol. 5, no. 2, pp. 163–179, 1984. [Online]. Available: https://doi.org/10.1016/0196-6774(84)90024-5

[25] J. Leskovec and J. Mcauley, in *Learning to discover social circles in ego networks*, 2012, pp. 548–556.

[26] J. Yang and J. Leskovec, “Defining and evaluating network communities based on ground-truth,” in *Proceedings of the ACM SIGKDD Workshop on Mining Data Semantics*, 2012, pp. 1–8.

[27] Y. Zhou, H. Cheng, and J. X. Yu, “Graph clustering based on structural/attribute similarities,” *Proceedings of the VLDB Endowment*, vol. 2, no. 1, pp. 718–729, 2009.

[28] M. E. Newman and D. J. Watts, “Renormalization group analysis of the small-world network model,” *Physics Letters A*, vol. 263, no. 4-6, pp. 341–346, 1999.

[29] R.-H. Li, J. Su, L. Qin, J. X. Yu, and Q. Dai, “Persistent community search in temporal networks,” in *Proceedings of the International Conference on Data Engineering (ICDE)*, 2018, pp. 797–808.

[30] Y. Fang, X. Huang, L. Qin, Y. Zhang, W. Zhang, R. Cheng, and X. Lin, “A survey of community search over big graphs,” *The VLDB Journal*, vol. 29, pp. 353–392, 2020.

[31] Y. Fang, Y. Yang, W. Zhang, X. Lin, and X. Cao, “Effective and efficient community search over large heterogeneous information networks,” *Proceedings of the VLDB Endowment*, vol. 13, no. 6, pp. 854–867, 2020.

[32] M. Sozio and A. Gionis, “The community-search problem and how to plan a successful cocktail party,” in *Proceedings of the International Conference on Knowledge Discovery and Data Mining*, 2010, pp. 939–948.

[33] W. Cui, Y. Xiao, H. Wang, and W. Wang, “Local search of communities in large graphs,” in *Proceedings of the International Conference on Management of Data (SIGMOD)*, 2014, pp. 991–1002.

[34] F. Bonchi, A. Khan, and L. Severini, “Distance-generalized core decomposition,” in *proceedings of the International Conference on Management of Data (SIGMOD)*, 2019, pp. 1006–1023.

[35] V. Batagelj and M. Zaversnik, “An o (m) algorithm for cores decomposition of networks,” *arXiv preprint cs/0310049*, 2003.

[36] Y. Zhang and J. X. Yu, “Unboundedness and efficiency of truss maintenance in evolving graphs,” in *Proceedings of the International Conference on Management of Data (SIGMOD)*, 2019, pp. 1024–1041.

[37] W. Cui, Y. Xiao, H. Wang, Y. Lu, and W. Wang, “Online search of overlapping communities,” in *Proceedings of the International Conference on Management of Data (SIGMOD)*, 2013, pp. 277–288.

[38] L. Yuan, L. Qin, W. Zhang, L. Chang, and J. Yang, “Index-based densest clique percolation community search in networks,” *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 30, no. 5, pp. 922–935, 2017.

[39] L. Chang, X. Lin, L. Qin, J. X. Yu, and W. Zhang, “Index-based optimal algorithms for computing steiner components with maximum connectivity,” in *Proceedings of the International Conference on Management of Data (SIGMOD)*, 2015, pp. 459–474.

[40] J. Hu, X. Wu, R. Cheng, S. Luo, and Y. Fang, “On minimal steiner maximum-connected subgraph queries,” *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 29, no. 11, pp. 2455–2469, 2017.

[41] B. W. Kernighan and S. Lin, “An efficient heuristic procedure for partitioning graphs,” *The Bell system technical journal*, vol. 49, no. 2, pp. 291–307, 1970.

[42] M. E. Newman, “Community detection and graph partitioning,” *Europhysics Letters*, vol. 103, no. 2, p. 28003, 2013.

[43] M. Girvan and M. E. Newman, “Community structure in social and biological networks,” *Proceedings of the National Academy of Sciences (PNAS)*, vol. 99, no. 12, pp. 7821–7826, 2002.

[44] A. Clauset, M. E. Newman, and C. Moore, “Finding community structure in very large networks,” *Physical Review E*, vol. 70, no. 6, p. 066111, 2004.

[45] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, “Fast unfolding of communities in large networks,” *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, p. P10008, 2008.

[46] D. Kempe, J. Kleinberg, and É. Tardos, “Maximizing the spread of influence through a social network,” in *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, 2003, pp. 137–146.

[47] C. Wang, W. Chen, and Y. Wang, “Scalable influence maximization for independent cascade model in large-scale social networks,” *Data Mining and Knowledge Discovery*, vol. 25, pp. 545–576, 2012.

[48] S. Chen, J. Fan, G. Li, J. Feng, K.-l. Tan, and J. Tang, “Online topic-aware influence maximization,” *Proceedings of the VLDB Endowment*, vol. 8, no. 6, pp. 666–677, 2015.

[49] Y. Li, D. Zhang, and K. Tan, “Real-time targeted influence maximization for online advertisements,” *Proc. VLDB Endow.*, vol. 8, no. 10, pp. 1070–1081, 2015.