Keyword Aware Influential Community Search in Large Attributed Graphs

Md. Saiful Islam, Mohammed Eunus Ali, Yong-Bin Kang, Timos Sellis, and Farhana M. Choudhury

Abstract—Influential community search (ICS) on a graph finds closely connected groups of vertices that dominate other such groups in the graph. The ICS has many applications in recommendations, event organization, and so on. In this paper, we introduce a new variant of ICS, namely Keyword-aware Influential Community Query (KICQ), that finds communities with the highest influential scores and whose keywords match with the query terms (a set of keywords) and predicates (AND or OR). The KICQ enables a user to find influential communities of her interest by specifying a set of keywords. It is challenging to find such communities from large graphs as the traditional pre-computation approaches are not applicable due to the change of query terms at every instance of the search. To solve this problem, we design two efficient algorithms: (i) a branch-and-bound approach that exploits the bounds computed from already explored communities to prune the search space, and (ii) a novel index-based approach that hierarchically organizes candidate communities and keywords with associated bounds to quickly identify the desired communities. We propose a novel influence measure for a community that considers both the cohesiveness and influence of the community and eliminates the need for specifying values of internal parameters of a network. Finally, we present detailed experiments and a case study to demonstrate the effectiveness and efficiency of the proposed approaches.

Index Terms—Influential community search, semantic keyword, community search in attributed graph, influence, social network.

1 Introduction

Finding communities from large networks has received significant attention in recent years due to its diverse practical applications that include event organization [34], friend recommendation [29], and e-commerce advertisement [21]. Traditionally, community search (CS) on a large network involves finding a community around a given query vertex that satisfies *query parameters* like *connectivity* and *cohesiveness constraints* [13], [14], [18], [34]. More recent research works [22], [24] have focused on finding *influential communities* from a network where the common goal is to find a closely connected group of users (vertices) who have some *dominance* over other users in the network. We suggest that the influence measure of a community is subjective, and depends on the interests of the query issuer. For example, a popular football community has little or no influence over a user not interested in sports. Thus, it is important to incorporate the user's interests while searching for influential communities, which is ignored in the current literature.

Though a large body of research have already addressed various aspects of CS, in this paper, we address the following gaps in the existing works:

First, traditional CS works on an attributed graph find a group of neighboring vertices whose keywords have high similarity with the given query vertex keywords and satisfy the required structural constraints [14], [18] (e.g., C_1 in Figure 1 with n_3 as the query vertex, and parameter k = 4 where k-truss is the structural constraint). A major limitation of such CS techniques is that the user needs to define the query vertex and the structural properties of the community explicitly, which might not be suitable in many application domains. A couple of recent studies [4], [42] tried to address these limitations by finding cohesive (i.e., k-core or triangle density) communities having close similarity with query keywords. However, they do not consider the influence of individuals in different keywords (e.g., C_1 in Figure 1 is highly cohesive in terms of structure and keyword, but two highly influential vertices n_1, n_2 are ignored in this case) and also do not support flexible conjoining (e.g., AND or OR predicates) of query keywords. **Second**, existing works on influential community search only work on *non-attributed* graphs and also require specific values of structural parameters. For example, [24] requires users to mention the value of k while finding k-core based communities; similarly, [22] requires the values of the minimum number of vertices m in a community and the maximum distance p between any two vertices while finding an mp-clique based community. Although such parameters allow high customization in search, we argue that the choice of these parameters highly depends on the internal structure of the graph in practice. For example, if k

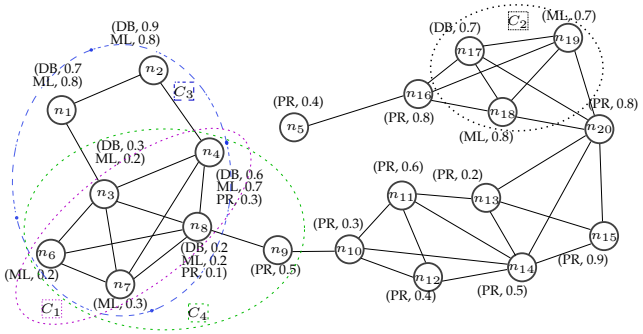


Fig. 1: An attributed author-author graph, where each vertex has an associated list of attributes (keywords) and influences denoting her expertise. Different types of CS communities are marked as $C_1 - C_4$.

- *Md. Saiful Islam and Mohammed Eunus Ali are with the Department of CSE, BUET, Bangladesh.* ***E-mail:*** *saifulislam@cse.buet.ac.bd, eunus@cse.buet.ac.bd*
- *Yong-Bin Kang and Timos Sellis are with Swinburne University of Technology, Australia.* ***E-mail:*** *ykang@swin.edu.au, tsellis@swin.edu.au*
- *Farhana M. Choudhury is with Melbourne University, Australia.* ***E-mail:*** *farhana.choudhury@unimelb.edu.au*

Manuscript received April 19, 2005; revised August 26, 2015.

is set to a high value (e.g., 4) in a small graph (Figure 1), no community is returned by [24] because there is no 4-core in this graph; and if k is low (e.g., 2), the community returned (e.g., C_2 in Figure 1) does not have high cohesiveness. Similarly, given a query vertex, [22] only returns the desired community under specific constraints of parameter values (e.g., C_4 in Figure 1 with parameters $m \leq 6$ and $p = 2$), which is impractical for an external user. Thus flexibility in parameters while searching for influential communities is crucial. **Third**, the influence of a community is defined as the minimum influence among all members in [24]; thus, a member with low influence can severely affect the influence of a community.

To fill the above research gaps and to support a new set of applications, we propose a novel parameter-free influential CS query, namely Top-r Keyword-aware Influential Community Query ($KICQ$), on an attributed graph, where vertices (i.e., users) are augmented with attributes (i.e., keywords). To motivate the application scenarios of the KICQ, let us consider the following applications: **(i)** A prospective Ph.D. student may want to search for strong (i.e., influential) research groups in her areas of interest from a research network. Figure 1 presents such a scenario. Here, vertices $n_1 - n_{20}$ are authors who published papers in field of studies relevant to "Machine Learning (ML)", "Database (DB)", and "Pattern Recognition (PR)". The student may be interested in finding the most influential community who are working in "ML" or "DB." The $KICQ$ returns the community C_3 as shown in Figure 1 as the most influential community (see Section 5 for the influential metric) since the members of the community have influence in either "ML" or "DB", the community is dense and also contains highly influential members. **(ii)** In the context of traditional social networks such as Twitter, we can represent the network as an attributed graph, where a user is connected with her friends/followers, and her topics of interest can be extracted from her posts/likes/shared contents. From such a graph, one may want to find the most influential communities for given keywords like "music" or "movie". **(iii)** Similarly, in the location-based social networks, one can find influential tourist groups who frequently visit a set of locations, e.g., "Rome" and "Milan".

A major challenge in realizing such a query (i.e., $KICQ$) comes from the fact that communities and their influences need to be computed and compared on the fly based on the set of keywords in the query, and thus existing pre-computation based approaches are not suitable for our purpose. Our major contributions are the following:

First, we design $KICQ$ in such a way that enables users to issue an influential community search query intuitively by merely using a set of query terms (words or phrases), and predicates (AND or OR) (addressing the first limitation). In this context, we propose a novel word-embedding based keyword similarity model that enables *semantic community search*, which substantially alleviates the limitations of *exact keyword* based community search. **Second**, we propose a new influence measure for a community that considers both the cohesiveness and influence of the community and eliminates the need for specifying values of internal parameters of a network (addressing the second limitation). We propose an influence measure that captures the influence in a better intuitive sense rather than the influence of the community being dominated by the minimum influence of a member (addressing the third limitation). (Section 5) **Third**, we solve the problem of keyword-aware influential community search by proposing two efficient techniques: (i) branch-and-bound based pruning that uses the derived bounds from already explored communities to effectively prune the communities that cannot be a part of the answer set, and (ii) an index-based approach, where we design an index, `KIC-tree`, to organize the keywords, communities, and related lower and upper influence score bounds, to quickly find the desired communities from a large scale attributed graph. (Section 6) **Fourth**, we conduct comprehensive experiments and a case study to show that our algorithms are highly efficient and effective in retrieving keyword aware influential communities. We share the datasets and additional technical details to help future works in this direction. (Section 7)

2 Related Works

Finding communities from a large graph has been an engaging research direction for a long time. Although the definition of 'community' varies among different studies, cohesive subgraphs (e.g. such as maximal cliques [8], k-core [7], and k-truss [36]) form the basis of modeling communities. The task of finding communities can be divided into two major classes: community detection (CD), and community search (CS).

In CS, communities are defined based on a query, and CS solutions aim to efficiently find communities given a query in an online manner. CD methods usually use global criteria (e.g., cohesiveness) to detect all the communities from an entire graph, where the focus is more on quality than efficiency. Link based analysis was popular in initial CD studies [16] that do not consider attributes in a graph. Clustering based techniques [19], [31], [39], [44], and topic modeling [26], [32] are used in recent studies to detect communities in attributed graphs. However, none of the CD studies enables a user to find specific communities of her interest, which is the main focus of our work.

Referring to Table 1, the basic CS works on non-attributed graphs focus on finding communities from the graph that contain vertices given in the query. These studies mostly focus on the connectivity among the members and do not consider the influence of individuals. The notion of influential CS (on non-attributed graphs) was introduced by [24], where vertices are assigned an influence score, and the influence of a community is modeled as the minimum influence of the members. Later works [5] and [3] develop faster algorithms to solve the influential CS problem. Also, [43] studies influential CS in an undirected weighted graph, where the weight of an edge represents the semantic intimacy between two vertices. A more recent work [22] defines a community in terms of kr-clique and designs algorithms to retrieve the most influential community. However, all of these studies ignore rich information of vertices (e.g., keywords denoting expertise of users) found in attributed graphs. Additionally, these studies require several vertices or internal parameters as part of a query, which is very difficult for a user who does not have enough knowledge of the graph.

Domain	Non-attributed graph	Attributed graph	
		Keyword	Others
Basic CS	[1], [10], [17], [25], [34], [40], [41]	[4], [9], [14], [18], [20], [42]	[6], [13], [37], [45]
Influential CS	[3], [5], [22], [24], [43]	-	[23]

TABLE 1: Existing community search works

There are several CS works on attributed graphs that do not consider the influence of users. For example, [14] proposes the ACQ algorithm to find subgraphs satisfying structural and keyword cohesiveness; [18] explores attribute driven CS in terms of k-truss; and [6] studies CS in an attributed graph where each vertex has a *profile* consisting of a set of keywords arranged in a tree structure. Another work [9] employs keyword search techniques to facilitate CS in attributed graphs. However, these studies also require a set of vertices and/or internal parameters as part of the query. Few recent works [4], [42], [42] study keyword-based CS that take a set of keywords as input and return a subgraph as the community that has the *best* match with the given set of query keywords. In these works, the cohesiveness of the subgraph is measured differently, i.e., k-core in [42], triangle density in [4], and average proximity in [20]. To decide a single best-matched subgraph, they define functions that consider the presence or absence of keywords and structural cohesiveness in the subgraph. These differ ours as they only consider the presence or absence of keywords in different vertices of the subgraph and cannot be adapted for the scenario where we need to rank the communities and each vertex has a certain degree of influence in each keyword.

A recent work [23] studies skyline community search where each vertex is associated with a d-dimensional influence score. However, their study is designed for low values of d (i.e., $d < 5$). With $d = 5$, their algorithms require more than 10^3 seconds in a graph with half a million vertices. This study cannot be extended for an attributed graph where vertices are associated with influence scores in multiple keywords, because there can be millions of keywords (dimension) in such an attributed graph. Also, this approach cannot search communities of specific interest.

In this study, we aim to solve the influential CS problem in an attributed graph, where vertices are attributed with keywords denoting their areas of expertise, and each vertex is associated with an influence score for every keyword. To the best of our knowledge, this is the first influential CS study that allows an external user to provide her areas of interest in terms of a set of keywords and a predicate (AND/OR), and retrieves the most influential communities based on these keywords.

3 Problem Definition and System Overview

To present the overview of the system, we first present some important definitions:

- *attributed graph*: An attributed graph $G^+(V, E, A)$ is an undirected graph, where V is the set of vertices and E is the set of edges. Each vertex v is associated with a set of tuples of the form $A_v = \{(w_i, s_v(w_i))\}$, where w_i is a keyword and $s_v(w_i) \in [0, 1]$ is the influence score of vertex v in keyword w_i.
- *maximal k-core*: Let H be a subgraph of G^+, induced by the set of vertices $V_H \subseteq V$. Let the degree of a vertex v in H is denoted by $deg_H(v)$. H is a k-core if $\forall_{v \in V_H} deg_H(v) \geq k$, where k is a non-negative integer. H is a maximal k-core if there is no super k-core in G^+ that contains H. We consider the connected components of maximal k-cores as the *influential communities*, where the *influence* is defined as Equation 2 (see Section 5). k is termed as *cohesion factor* in this paper.
- $KICQ$: $KICQ$ is the keyword-aware influential community query. Let $G^+(V, E, A)$ be an attributed graph, $q(T, P)$ be a query tuple where $T = \{t_1, t_2, \cdots, t_n\}$ is a set of terms (i.e., words or phrases) and P is a predicate (AND, OR) for conjoining the query terms. Let, k_{min} be the minimum cohesion factor for being a candidate community, and r be a positive integer specifying the number of top communities to be returned. Then, we form the $KICQ$ as a quadruple (X, P, r, k_{min}), where $X = \{X_{t_1}, X_{t_2}, \cdots, X_{t_n}\}$ and X_{t_i} is the set of semantic keywords (see Section 4.1) of term t_i. $KICQ$ finds r *most influential communities* $H_1, H_2, ..., H_r$ from G^+.

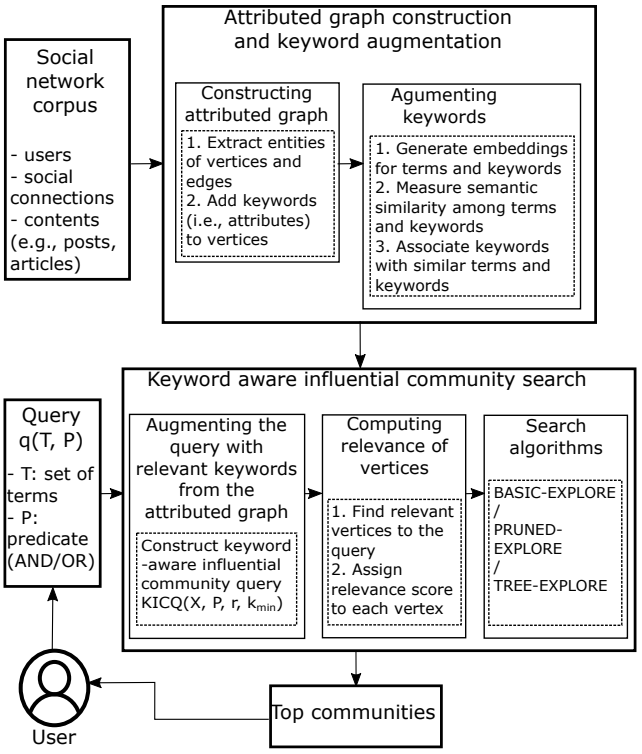


Fig. 2: The overview of our system

Our system overview is presented in Figure 2. It is mainly divided into two phases. First, we construct a keyword-aware attributed graph from a social network corpus. Second, we focus on searching keyword-aware influential communities using the constructed attributed graph, given a query as a set of terms and predicates.

The distinctive features of the first phase are as follows: (1) We build an attributed graph from a domain corpus by extracting entities of possible vertices and edges to represent the social network. Each user is associated with a set of keywords based on the contents of the user in the network.

(2) To enable keyword-aware influential community search, we augment keywords with their semantically related terms and keywords using *word embedding vectors* [27] as an external knowledge source. The output of this step is a graph, called *keyword-aware attributed graph*.

We briefly describe the second phase as follows: (1) Initially, a user raises query $q(T, P)$ consisting of a set of query terms T, and a predicate P. The terms T need to match with the keywords in the attributed graph to find meaningful communities. We acknowledge the difficulty faced by the users to put the exact terms while raising a query, and therefore, augment each query term with a semantically meaningful set of keywords. The output of this step is a $KICQ$. (2) Given a $KICQ$ query, we find the relevance scores of vertices to the query (Equation 1), which are used to compute the scores of potential influential communities. We argue that a measure that rewards both the cohesiveness of the community and the high influence of the members, and does not require user input of any internal parameters (e.g., k in k-core) is more preferable than the existing influence measures. We propose a linear weighted summation of the cohesiveness of the community and the total influence of the members of the community to estimate the overall score of a community (Section 5). (3) Given the augmented query and the influential score function, our focus is now to retrieve top-r most influential communities relevant to the query. Since we are the first to propose the keyword-aware influential community search problem, and existing pre-computation based approaches are not suitable to retrieve communities for any given query, we first present a baseline solution named `BASIC-EXPLORE` followed by two novel efficient algorithms: `PRUNED-EXPLORE` and `TREE-EXPLORE`.

4 KEYWORD AUGMENTATION

In a social network, one's expertise can be represented by a collection of terms (words or phrases). There are millions of such terms in a large network and it is difficult for users to come out with the exact terms while raising a query. We propose a semantic keyword similarity model that can augment any term with relevant keywords. This model is used to extend the keywords in the attributed graph, and associate appropriate keywords for each term in the query.

4.1 Semantic keyword similarity model

Finding semantically related keywords of a term is not trivial. Two or multiple terms with slight syntactic difference can indicate the same keyword (e.g., "error detection and error correction" and "error detection and correction"). Even two different terms can represent the same keyword (e.g., "AI" and "artificial intelligence") or can be semantically similar (e.g., "neural network" and "deep learning").

We adopt Word2Vec [27] model to generate a vector (semantic representation) of any given word. We train this model with a domain corpus after stopword removal, tokenization, and lemmatization [30]. If keyword/term contains multiple words, the representative vector is formed using the average of the embedding vectors of the constituent words. Now, given any two terms t_1 and t_2, we denote their embedding vectors as x_{t_1} and x_{t_2}, respectively. To estimate a similarity between these two terms, denoted as $S(t_1, t_2)$, we can use widely used `cosine` similarity of their embedding vectors x_{t_1}, x_{t_2} [28]. We also propose a new similarity, which yields an attributed graph of better quality:

indirect cosine: Given a term t, its vector V^t is denoted as $V^t = [(w_1^t, s_1^t), (w_2^t, s_2^t), \cdots, (w_L^t, s_L^t)]$, where w_i^t is the i^{th} most similar word to x_t, s_i^t is the corresponding similarity score, and L is the number of similar terms of t.

Given two terms t_1 and t_2, we construct a vocabulary, U combining words of V^{t_1}, and V^{t_2}. Formally, $U = \{w : (w, s) \in V^{t_1}$ or $(w, s) \in V^{t_2}\}$. To simplify our notation, let $U = \{w_1, w_2, \cdots, w_n\}$, where $n = |U|$. Now, we define another vector $SV^t = [s_1, s_2, \cdots, s_n]$, where s_i is the similarity score of term t to word $w_i \in U$, which can be found from V^t. If $(w_i, s_i) \notin V^t$, s_i is set to 0. Finally, $S(t_1, t_2)$ is calculated as the cosine similarity of SV^{t_1} and SV^{t_2}.

Finally, for any term t, we calculate its similarity with all the keywords in the given attributed graph and find M-top most relevant keywords X_t ranked based on the similarity scores. M is a system configurable parameter. By default, M is set to 10.

4.2 Augmenting keywords for KICQ in attributed graph

In attributed graph G^+, a vertex v is associated with a set of keywords. For all vertices, we extend each keyword t with its M-top most relevant keywords X_t using the semantic similarity model. For any keyword $w \in X_t$, we model the influence score of vertex v, $s_v(w) = s_v(t)$ since w and t are semantically similar.

A query $q(T, P)$ consists of a set of terms $T = \{t_1, t_2, \cdots, t_n\}$ and a predicate P. First, the semantic similarity model is used to augment each term t_i with the set of relevant keywords X_{t_i}. Then the system parameters r and k_{min} are used to formulate the keyword aware influential community query, $KICQ(X, P, r, k_{min})$ where $X = \{X_{t_1}, X_{t_2}, \cdots, X_{t_n}\}$.

5 INFLUENTIAL COMMUNITY MEASURES

We design a scoring function that considers connectivity, cohesiveness, influence of individuals and the community size, and assigns a score for ranking the candidate communities given a query.

First, for a given query, we redefine the influence of a vertex based on its relevance to the query. The query relevance score γ_v of a vertex v is estimated as follows: each vertex v in the attributed graph is annotated with keywords and their influence score for the corresponding keywords, i.e., $(w_i, s_v(w_i))$. To estimate the relevance score, $\gamma_v \in [0, 1]$, we need to consider the list of semantic keywords X and the predicate P in the $KICQ$ query. Formally, we use the following definition for computing γ_v:

$$\gamma_v = \mathbf{f}_{X_{t_i} \in X}[\mathbf{g}_{w \in X_{t_i}} s_v(w)] \quad (1)$$

Here, **f** and **g** are two aggregate functions: **g** combines the relevance of the vertex for the semantic keywords of a query term, and **f** combines the relevance scores in all terms considering the predicate P. We use **g** as the aggregate function, *MAX*; whereas we use **f** as *MIN* for AND predicate and *MAX* for OR predicate, respectively. *MIN* aggregate ensures that a vertex has high relevance to all the terms, while *MAX* only requires high relevance to any of the terms.

Now, we use a linear weighted summation of the cohesiveness and influences to calculate the overall score of a community. Let $H = (V_H, E_H)$ is a subgraph of attributed graph $G^+(V, E, A)$. If H is a community (connected component of maximal k-core), then the score of H is:

$$\zeta(H) = \beta \times \underbrace{\frac{k}{max\text{-}deg(G^+)}}_{\text{Cohesiveness score}} + (1-\beta) \times \underbrace{\frac{\sum_{v \in V_H} \gamma_v}{|V|}}_{\text{Influence score}} \quad (2)$$

Here, $max\text{-}deg(G^+)$ is the maximum degree of all vertices in G^+. Both the cohesiveness and the influence score of a community are normalized within $[0, 1]$, and the preference parameter $\beta \in [0, 1]$ defines the importance of one score relative to the other.

Since we model a community using connected k-core, connectivity, and cohesiveness are ensured. $max\text{-}deg(G^+)$ and $|V|$ is constant for attributed graph G^+. Thus the influence score of community H depends on the sum of influential scores $\sum_{v \in V_H} \gamma_v$ which prefers large communities with highly influential individuals. Although it may allow some low influential nodes to be included, as long as these low influential members do not disrupt the cohesiveness of the community, the score of this community is not penalized, which is the case in [24]. We acknowledge that such a measure is not unique, and other measures can be explored in the future. However, experiments and the case study using real datasets (presented in Section 7) demonstrate that our proposed influence measure can capture cohesive communities with highly influential members.

6 ALGORITHMS

Existing studies on influential CS [22], [24] focus on non-attributed graphs and thus do not allow users to customize the community search using a set of keywords. If we want to modify these algorithms to answer the KICQ, one possible way is to pre-compute communities for all possible combinations of keywords, which is impractical due to an exponential number of keyword combinations. Moreover, for each combination, the notion of *influential community* changes.

In this section, we present algorithms for finding r most influential communities from the attributed graph G^+ for a given $KICQ(X, P, r, k_{min})$. We first provide a basic solution, `BASIC-EXPLORE`, followed by our first branch-and-bound algorithm, `PRUNED-EXPLORE`. Finally, we design a novel index (`KIC-tree`) that directs us towards a novel, efficient, and scalable online search algorithm, `TREE-EXPLORE`.

6.1 A straightforward approach, `BASIC-EXPLORE`

A straightforward approach to answer $KICQ$ on a large graph is as follows. First, we extract the subgraph, which we call the *query essential subgraph*, G_q, containing vertices and edges that are relevant to the query. Then we find all the connected components of maximal k-core subgraphs for all possible values of k. Finally, we return the top r communities having the highest influential community scores as per Equation 2.

Finding G_q**.** The query essential subgraph, $G_q(V_q, E_q, \gamma)$ is a subgraph of the attributed graph $G^+(V, E, A)$ induced by V_q, the set of vertices with a non-zero query relevance score. In G_q, each vertex v is annotated with its relevance score γ_v, and E_q is the set of edges between any two vertices in V_q. To efficiently generate the G_q, we maintain an inverted index, where for each keyword w, a list IL_w of the vertices that contain w is stored. Thus, for a given $KICQ$ query, V_q can be obtained by,

$$V_q = \begin{cases} \bigcap_{X_{t_i} \in X} [\bigcup_{w \in X_{t_i}} IL_w], & \text{if } P = AND \\ \bigcup_{X_{t_i} \in X} [\bigcup_{w \in X_{t_i}} IL_w], & \text{otherwise} \end{cases} \quad (3)$$

After retrieving V_q, we compute the query relevance score of each vertex $v \in V_q$ (Equation 1) and retrieve E_q that denotes the connections between all pairs of vertices in V_q.

Finding k***-cores and most influential communities.*** First, we compute core decomposition for all vertices in G_q using the $O(|E_q|)$ algorithm proposed by Batagelj et al. [2]. A priority queue Q is used to hold our solution. We initialize Q with r empty communities having 0 score. In our case, a community must be at least k_{min}-core. Again, the maximum cohesion factor of a community in G_q can be $max\text{-}deg(G_q)$, since there is no vertex in G_q with a higher degree. Thus we need to first find all connected components of maximal k-cores from G_q, where the value of k is in range $[k_{min}, max\text{-}deg(G_q)]$. Then, we compute the influential scores of each computed community, and finally, maintain the top-r communities in Q ordered by the scores of the communities.

Time complexity. Finding the relevance of vertices can be done in $O(|V_q| \times N_w)$ time, where $N_w = \sum_{X_i \in X} |X_i|$. If the graph is implemented with adjacency list, E_q can be obtained in $O(|V_q|)$ time. So, time complexity for computing G_q is $O(|V_q| \times N_w)$. Core decomposition of V_q is done in $O(|E_q|)$ time. Exploring a maximal k-core requires computing its connected components ($O(|V_q| + |E_q|)$), obtaining k-core vertices ($O(|V_q|)$), and computing scores of each connected components ($O(|V_q|)$). The cost of queue maintenance is inferior compared to other components. So, if N_k is the number of candidate communities with cohesion factor k, then the runtime of exploring all k-cores is bounded by $O(max\text{-}deg(G_q) \times (|V_q| + |E_q| + N_k \times |V_q|)$, that can be simplified as $O(max\text{-}deg(G_q) \times |V_q|^2)$ for a dense graph [1]. Since, this dominates the time complexity of finding G_q, the overall complexity of `BASIC-EXPLORE` is $\mathbf{O(max\text{-}deg(G_q) \times |V_q|^2)}$.

6.2 Pruned exploration approach, `PRUNED-EXPLORE`

The major bottleneck of `BASIC-EXPLORE` is that it needs to explore all maximal k-cores, for different values of k, and find the connected components of each maximal k-core subgraph. Such exploration is computationally expensive for a large graph. Instead of directly exploring the subgraphs to compute the maximal k-core and its connected components (communities), we first estimate the upper bound score of the communities of the corresponding

1. $N_k < |V_q|$ and for any dense graph $G(V, E)$, $|E|$ is $O(|V|^2)$

Algorithm 1 PRUNED-EXPLORE (H, k)

Input: Subgraph of G_q $H = (V_H, E_H)$, cohesion factor k.
1: $k = \min(k, min\text{-}deg(H))$
2: CC^k = connected components of maximal k-core in H
3: **for** all $h \in CC^k$ **do**
4: **if** score of h, $\zeta(h) > r^{th}$ best score **then**
5: remove the last item and insert h in Q
6: **for** $k' = k + 1$ to $max\text{-}deg(G_q)$ **do**
7: **if** upper bound score, $\zeta^*_{k'}(h) > r^{th}$ best score **then**
8: PRUNED-EXPLORE(h, k')
9: break

subgraph. This bound can be used to prune a large number of redundant subgraphs that cannot be a part of the top-r influential communities.

First, we find the query essential subgraph G_q, compute core decomposition, and initialize priority queue Q as described in Section 6.1. Now, we need to explore G_q to retrieve communities for all possible values of k. As discussed before, the value of k must be between k_{min} and $max\text{-}deg(G_q)$. We propose the following lemmas, which pave the foundation of our pruning.

Lemma 1. Let, $H(V_H, E_H)$ be a subgraph of G_q. For any community in H, the maximum influence score can be the sum of the query relevance scores of all vertices in H. Thus, without computing the vertices of k-core subgraph, we can calculate the upper bound of the score of any community in H for a particular value of k as follows.

$$\zeta^*_k(H) = \beta \times \frac{k}{max\text{-}deg(G^+)} + (1-\beta) \times \frac{\sum_{v \in V_H} \gamma_v}{|V|} \quad (4)$$

Lemma 2. If $H = (V_H, E_H)$ is a subgraph and $min\text{-}deg(H) = \min_{v \in V_H}(deg_H(v)) > k$, then any community in H must be at least $min\text{-}deg(H)$-core.

According to Lemma 1, we can prune a subgraph if its upper bound score is lower than the r^{th} best score of already retrieved communities from G_q. Moreover, Lemma 2 helps us to avoid the computation of certain cores from G_q.

Now, we develop a recursive procedure PRUNED-EXPLORE to search for influential communities in G_q. Algorithm 1 outlines the procedure. Initially, PRUNED-EXPLORE(G_q, k_{min}) is called to extract communities with minimum cohesion factor. In later steps, the procedure is recursively called to extract communities with higher cohesion factors.

Let us consider that we want to find communities with cohesion factor k, from a subgraph $H(V_H, E_H)$ of G_q. In line 1, we determine the minimum degree of the vertices in H, $min\text{-}deg(H)$. If $min\text{-}deg(H) > k$, we set $k = min\text{-}deg(H)$ and directly compute such k-cores (according to Lemma 2). In line 2, we find the set of connected components of maximal k core of H, denoted by CC^k. The loop in line 3 runs for each connected component. We update the priority queue if any connected component's score is higher than the current top-r communities in lines 4-5. We explore the connected component for higher values of k in lines 6-9. We use Lemma 1 to prune exploration for the values of k for which the upper bound of the score is lower than the r^{th} best community. When the procedure terminates, the queue holds the final top-r communities.

6.3 Keyword indexed tree exploration, TREE-EXPLORE

Though the above PRUNED-EXPLORE is efficient, it still explores subgraphs and their connected components with low cohesiveness, which usually do not contain the most influential communities. This exploration can be costly, especially in a scenario where the query essential graph, G_q, turns out to be very large. So, we propose a novel index, namely *keyword indexed core-label tree* (KIC-tree), that pre-computes and organizes the connected components of maximal k-core subgraphs hierarchically with computed upper bound of influence scores for each keyword.

The key idea of our KIC-tree based $KICQ$ comes from the following observations:

(i) Top communities are structurally cohesive and thereby can be retrieved by exploring the subgraphs of higher cohesion factors. Thus, if k-cores are precomputed, disregarding the associated keywords, we can still prune the subgraphs with a low k value.

(ii) Communities are represented using connected maximal k-cores which are nested, i.e., by definition, a $(k+1)$-core is also a k-core ($k \geq 0$). This property helps to store all the connected components of maximal k-cores in compressed tree-based structures as shown in previous works ICP-index [24], CL-tree index [14].

(iii) We can compute the upper bounds for both the components: influence and cohesiveness, of the scoring function, and use these upper bounds to prune the search space during query time.

We first discuss the basic structure of the KIC-tree index. Then we present the upper bounds for individual keywords and aggregate them for a set of keywords and predicates (in $KICQ$) for an upper bound score of a node. We also show how the cohesiveness score can be bounded based on a pre-computed structure alone. Finally, we present our TREE-EXPLORE algorithm for influential community search using the KIC-tree. In this section, we use the term "node" to exclusively indicate a tree node.

6.3.1 KIC-tree index

The KIC-tree index organizes the connected components of k-cores into a space-efficient tree structure. We adopt the concept of the compressed tree-based structure of previous works (e.g., CL-tree index [14]), and augment the structure with derived bounds to prune the search space.

Figure 3 shows an example KIC-tree for the subgraph shown as the shaded region in Figure 1. The left shows the hierarchical representation of all maximal k-core connected components in the subgraph. We refer this tree as the *uncompressed tree*. The right figure shows KIC-tree index, a more compact representation of the left tree, which removes the graph vertices present in its descendant nodes ensuring that each graph vertex appears exactly once.

Let u be a KIC-tree node and $subtree(u)$ be the subtree rooted at u. The structure of u is as follows:

(i) k, the cohesion factor; (ii) $vertexSet$, the set of compressed graph vertices at node u; (iii) $childNodes$, the set of child nodes of u; (iv) k_{max}, the maximum cohesion factor of any connected component contained by the subtree(u); (v) $iList$, an inverted list containing the upper bounds of influence scores for all keywords appeared in subtree(u).

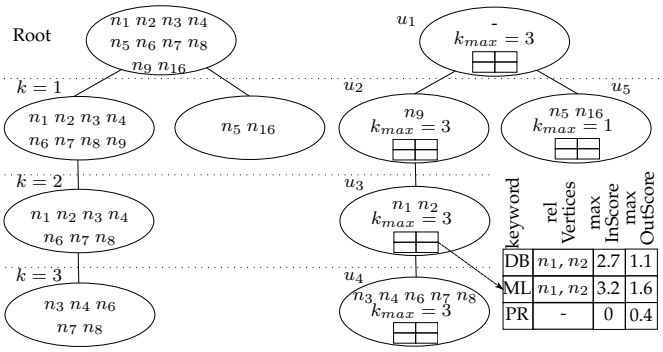


Fig. 3: KIC-tree for the subgraph (shaded) in Figure 1.

For each keyword w that appears in subtree(u), the inverted list $u.iList[w]$ contain the following elements:

(i) $relV$, a set of graph vertices in $u.vertexSet$ containing the keyword w;

(ii) $maxKNScore$, the upper bound of influence score component by only considering keyword w in a community (i.e., a connected component) contained by the $subtree(u)$, where the community must include at least one vertex present in node u containing the keyword w;

(iii) $maxKDScore$, the upper bound of influence score component by only considering keyword w of a community contained by the subtree(u), where the community does not include any vertex from $u.vertexSet$ (i.e., all vertices of the community come from the descendent nodes of u).

We compute $maxKNScore$ and $maxKDScore$ as follows.

For a node u, let $childV$ be the set of graph vertices stored at the descendent nodes of u, and $allV$ be the set of graph vertices at $subtree(u)$ (i.e., all the vertices in node u and its descendent nodes). If u is a leaf node, $u.childV = \emptyset$. Otherwise, $u.childV = \bigcup_{p \in u.childNodes} p.vertexSet$. On the other hand, in all cases, $u.allV = u.vertexSet \bigcup u.childV$.

Now, if there is no relevant graph vertex in node u for keyword w, then we set $maxKNScore$ as 0. Otherwise, the upper bound is the sum of influence scores of all graph vertices in $u.allV$. Formally,

$$u.iList[w].maxKNScore = \begin{cases} 0, \text{if } u.iList[w].relV = \emptyset \\ \sum_{v \in u.allV}(s_v(w)), \text{otherwise} \end{cases}$$

Here, $s_v(w) =$ influence score of vertex v for keyword w.

Now, $maxKDScore$ is the maximum influence score component among the communities represented by the descendant nodes of u. $u.iList[w].maxKDScore = 0$ if u is a leaf node. Otherwise, we can use the computed values of $maxKNScore$ to compute the $maxKDScore$ as follows.

$$u.iList[w].maxKDScore = \max_{p \in u.childNodes} p.iList[w].maxKNScore$$

Figure 3 (right) shows an example tree, where the table inside the ellipse represents the $iList$ of the corresponding node. For simplicity, we only show the $iList$ for node u_3.

6.3.2 Complexity analysis for index construction:

We use the `advanced` method proposed by Fang et. al. [14] that compresses the tree and for each node u, computes $u.iList[w].relV$ for all the relevant keywords of u. The time complexity of this method is $O(|E| \times \alpha(|V|))$, where *inverse Ackermann function*, $\alpha(|V|) < 5$ for all remotely practical values of $|V|$. For each $iList[w]$ entry, we also need to compute the two upper bounds $maxKNScore$ and $maxKDScore$. If A_{max} is the maximum number of keywords associated with a graph vertex, the time complexity for computing $maxKNScore$ is $O(A_{max} \times |V|)$. Computing $maxKDScore$ for a node u only requires visiting its $childNodes$ which is non-dominant. So, overall time complexity for index construction is $\mathbf{O}(|\mathbf{E}| \times \alpha(|\mathbf{V}|) + \mathbf{A_{max}} \times |\mathbf{V}|)$.

In $iList$, we need additional space to store two upper bound scores (constant space) for each keyword. The space cost is still dominated by storing $relVertices$ in $iList$. So, the space complexity remains $\mathbf{O}(\bar{\mathbf{A}} \times |\mathbf{V}|)$ as in [14], which is proportional to the graph size.

6.3.3 Computing upper bound scores for a query

Given a $KICQ(X, P, r, k_{min})$ query, we need to compute an upper bound influence score of a community denoted by S_{inf} and the maximum possible cohesiveness score of that community, S_k by using the precomputed upper bounds in `KIC-tree`. Then the upper bound of the total score of that community can be computed as $maxScore = \beta \times S_k + (1 - \beta) \times S_{inf}$ (as in Equation 2).

We define two upper bounds for the communities inside $subtree(u)$: (i) $maxNodeScore$, the maximum possible score of any community that can be exclusively found by exploring the connected k-core stored at node u and (ii) $maxDesScore$, the maximum possible score of any community that can be found by exploring the descendant nodes of u.

Computing $maxNodeScore$**:** For any community contained exclusively by node u, there must be at least one vertex v that is stored at u. Now, for any vertex v exclusive to node u, $u.k$ is the maximum core number. So, a subgraph containing v can be at most $u.k$-core (irrespective of any keyword) and the upper bound of cohesiveness score of any community contained by the node can be computed as $S_k = u.k/max\text{-}deg(G^+)$.

Now, for each keyword w, $u.iList[w].maxKNScore$ defines the upper bound of influence score component for any community in the subgraph exclusively contained by node u (considering the single keyword w). We combine these bounds considering all the keywords in the $KICQ$ query and compute the maximum influence score as:

$S_{inf} = \frac{1}{|V|} \times \mathbf{F}_{X_{t_i} \in X}(\sum_{w \in X_{t_i}} u.iList[w].maxKNScore)$

Here, **F** is an aggregate function that combines the influence scores of the community for multiple terms depending on the predicate P and division by $|V|$ normalizes the score within [0,1]. For the queries with OR predicate, a top community can be formed by joining multiple communities pre-computed for a single term, and these communities may have disjoint vertex set. So, it is safe to consider **F** as a **sum** aggregate. For the same reason, $\sum$ is explicitly used to combine the semantic keywords of a term. Again, for AND predicate, any graph vertex forming a community for a single term must be present in communities of other terms as well. So, **F** can be safely considered as **min** aggregate.

Computing $maxDesScore$**:** For any community contained by the descendant nodes of u, the maximum

Algorithm 2 TREE-EXPLORE (u, U)

Input: Tree node u, query relevant nodes U.
1: **if** u is an internal node **then**
2: Compute S_{inf} and S_k to compute $u.maxDesScore$
3: **if** $S_{inf} > 0$ and $u.maxDesScore \geq r^{th}$ best score **then**
4: **for** each $p \in (u.childNodes \cap U)$ **do**
5: TREE-EXPLORE(p, U)
6: **if** $u.k < k_{min}$ **then return**
7: Compute S_{inf} and S_k to compute $u.maxNodeScore$
8: **if** $S_{inf} > 0$ and $u.maxNodeScore \geq r^{th}$ best score **then**
9: $u.V_{rel}$ = relevant graph vertices in subtree(u)
10: Compute query relevance score of all vertices in $u.V_{rel}$
11: Compute $u.E_{rel}$, the edges among $u.V_{rel}$
12: Construct subgraph $H(u.V_{rel}, u.E_{rel})$, with each vertex annotated with relevance score
13: MODIFIED-PRUNED-EXPLORE(H, k_{min}, $u.k$)

cohesion factor is $u.k_{max}$ and the upper bound of cohesiveness score is $S_k = u.k_{max}/max\text{-}deg(G^+)$.

Again, for keyword w, $u.iList[w].maxKDScore$ already defines the upper bound of influence score component for any community contained by the descendant nodes. We combine these bounds for considering all the keywords in the $KICQ$ query and compute the maximum influence score similarly as computing $maxNodeScore$, i.e.,

$S_{inf} = \frac{1}{|V|} \times \mathbf{F}_{X_{t_i} \in X}(\sum_{w \in X_{t_i}} u.iList[w].maxKDScore)$

6.3.4 TREE-EXPLORE algorithm

We follow a *best-first* exploration strategy. Since the leaf nodes contain the communities with high cohesiveness while nodes near root contain communities with low cohesiveness, we explore the KIC-tree in a post-order manner. Likewise the previous exploration algorithms (e.g., PRUNED-EXPLORE), a priority queue Q initialized with r empty communities is used to store the results. The exploration algorithm, which we call TREE-EXPLORE is developed based on the following pruning techniques:

(i) Subtree pruning: For any node u, we examine the $u.maxDesScore$ before visiting its children. If it is less than the r^{th} best score, then none of the communities to be found in the descendent nodes can score higher than the current r^{th} top community. Therefore, we can skip visiting the descendant nodes of u.

(ii) Node pruning: Before exploring the pre-computed connected k-core subgraph at any node u, we examine the $u.maxNodeScore$. If it is less than the r^{th} best score, we can safely prune this exploration.

Algorithm 2 outlines the pseudocode for the KIC-tree traversal. The inverted list that we have used to find the G_q is adopted for computing U, the set of tree nodes relevant to a query. Initially, the recursive procedure TREE-EXPLORE(u, U) is called with u being the root of the KIC-tree.

For any internal node u, we first compute the influence score component S_{inf}, and the cohesiveness score component S_k of $u.maxDesScore$. If $S_{inf} = 0$, the descendants of u do not contain any relevant graph vertex, and therefore we do not need to visit subsequent nodes across the subtree. If $S_{inf} > 0$ and $u.maxDesScore >$ current r^{th} best score, then we visit its children (lines 1-5).

Now we explore the pre-computed connected k-core subgraph represented by node u. If the cohesion factor k in node u is less than k_{min}, we prune exploring the subgraph. Else, we compute the influence score component S_{inf} and the cohesiveness score component S_k of $u.maxNodeScore$. If $S_{inf} = 0$, then the node does not contain any relevant graph vertex, and we can safely skip exploring the subgraph. Again, we skip the exploration if $u.maxNodeScore$ is less than the current r^{th} best score.

If the exploration of the connected k-core subgraph cannot be pruned, we first need to find all the relevant graph vertices, $u.V_{rel}$ present in the subgraph. Since KIC-tree compresses these graph vertices by removing the ones present at descendant nodes, we need to decompress in a bottom-up manner. At any node u, the relevant graph vertices $u.V_{rel}$ can be computed like the vertices in QEG (Equation 3) just by replacing IL_w with $u.iList[w].relV$. For any internal node u, we need to add the relevant graph vertices in child nodes to $u.V_{rel}$. Then, we compute the relevance score of each vertex $v \in u.V_{rel}$ (Equation 1) and then compute the edges among these vertices, thereby construct the subgraph $H(u.V_{rel}, u.E_{rel})$ and explore it for top communities (lines 8-13).

The procedure MODIFIED-PRUNED-EXPLORE(H, k, k_{max}) is a slightly modified version of the procedure PRUNED-EXPLORE(H, k) that takes an extra argument k_{max}, the maximum value of cohesion factor for sub-graph H. Lines 6-9 in Algorithm 1 are replaced by the following:

6: **for** $k' = k + 1$ to k_{max} **do**
7: **if** $\varsigma^*_{k'}(h) > r^{th}$ best score **then**
8: MODIFIED-PRUNED-EXPLORE(h, k', k_{max})
9: break

Since no graph vertex at u belongs to any k-core with cohesion factor higher than $u.k$, here $k_{max} = u.k$. Initially, $k = k_{min}$. So, MODIFIED-PRUNED-EXPLORE$(H, k_{min}, u.k)$ is called to explore the subgraph H (line 17).

7 EXPERIMENTS

In this section, we present experiments on real datasets to evaluate our study. First, we discuss our experimental setup. Then, we evaluate the effectiveness of our proposed semantic similarity model. Finally, we evaluate the effectiveness, efficiency, and scalability of our proposed search algorithms, and present a case study to demonstrate the quality of the retrieved communities.

7.1 Experimental setup

We first discuss our experimental environment. Then, we describe the datasets followed by how the attributed graphs are generated, and queries are set up for the experiments. We also discuss the parameters that may affect the performance of our proposed algorithms. We have uploaded the constructed attributed graphs in a public repository[2]. The repository also contains a detailed description of datasets, query setup, and the semantic similarity model and its evaluation.

Our algorithms and all the compared community search algorithms are implemented in JAVA. Experiments were run on a virtual environment of OzSTAR[3] supercomputer with two cores of Intel Gold 6140 CPU 2.30 GHz, and 192 GB RAM. We assume that the graph and all the

2. https://github.com/saiful1105020/TKDE-2020-Additional-Contents
3. https://supercomputing.swin.edu.au/ozstar/

indexes will fit in the memory. For the simplicity of presentation, we use shorter names for our algorithms: BASIC, PRUNE, and TREE to represent BASIC−EXPLORE, PRUNED−EXPLORE, and TREE−EXPLORE respectively. For presenting any statistics of search algorithms, we use the average of 100 queries.

Datasets: We use three large real datasets (see Table 2) that reflect the real-life application scenarios. In OAG dataset, we model the authors as vertices and the co-authorship among them as edges. For Twitter and Gowalla datasets, users and followerships/friendships among them are modeled as vertices and edges.

Dataset	No. of vertices	No. of edges	No. of keywords
OAG [4]	1,000,000	15,677,940	1,000
Twitter	140,371	2,283,875	10,000
Gowalla [5]	407,533	2,209,169	2,727,464

TABLE 2: Datasets for experimental analysis

Attributed graph generation: We model keywords to represent the area of expertise of the users (e.g., "Database", "Music", "Mount Everest"). In OAG, author-provided keywords are captured as the keywords. Keywords in Twitter are extracted from user tweets by applying lemmatization [12]. In Gowalla, locations are considered as keywords. We apply our semantic similarity model to extend the keywords of OAG and Twitter datasets as mentioned in Section 4.1. The influence score of a user for a certain keyword is modeled as the user's percentile rank considering the number of citations (OAG)/ retweets (Twitter)/ check-ins (Gowalla). For example, an author having 0.98 influence score for "database" denotes that, the author has more citations than 98% of all authors considering the articles relevant to "database".

Query formulation: To generate a query for OAG and Twitter datasets, first, we choose a random number (1-5) of query terms from the most frequent $1,000$ keywords found in the attributed graph and then augment each keyword with its similar keywords using our semantic keyword similarity model. For Gowalla dataset, we randomly choose a set of locations within a range of 5 km as query terms. Unless specified otherwise, we generate 100 queries as mentioned above, and use the average to present any result.

Experiment parameters: The parameters shown in Table 3 affects the performance of our algorithms. We evaluate the effects of these parameters by varying one of them and keeping the others fixed at their default values.

Parameter	Range	Default
Dataset	OAG, Twitter, Gowalla	OAG
Size of default dataset (vertices)	$300K, 500K, 700K, 900K, 1M$	$500K$
Query predicate	AND, OR	OR
r	any integer within [1, 5]	3
k_{min}	any integer within [2, 50]	10

TABLE 3: Parameters for experimental analysis

7.2 Evaluation of semantic similarity model

We presented two approaches (cosine and indirect cosine) for finding semantic similarity between two terms or keywords. Here, we empirically evaluate which approach is the most effective. We use the 2012 ACM CCS[6] taxonomy that contains 2,113 topics and organizes them hierarchically based on relevance. Given such a taxonomy, [33] is a widely used approach to find the semantic similarity between any two topics (used as the ground truth). In our context, each topic in the taxonomy can be seen as a keyword or a term, and each of our proposed similarity measures can be considered as a ranker that finds the most similar topics to any topic in the taxonomy. So, we adopt Normalized Discounted Cumulative Gain ($NDCG$) [38], which is a popular performance measure of ranking systems. $NDCG$ gives more importance on correctly ranking a more relevant entity than entities with lower relevance. The goodness of ranking top-M entities is evaluated by $NDCG@M \in [0, 1]$ (1 denoting the perfect ranking, while 0 being the worst).

We obtain the vector representation of each topic in the taxonomy using Google's pre-trained word2vec model[7] that includes embedding vectors for a vocabulary of 3 million words and is trained on Google News dataset covering academic research. Table 4 shows the comparison among cosine and indirect cosine approaches in terms of $NDCG@50, NDCG@20, NDCG@10$. We choose the indirect cosine approach, as it outperforms cosine.

Metric	cosine	indirect cosine
NDCG@50	0.528	**0.542**
NDCG@20	0.537	**0.607**
NDCG@10	0.573	**0.633**

TABLE 4: Performance of the semantic similarity approaches

While using the indirect cosine approach, there is a challenge in determining a good value for L (see Section 4.1). To examine this, we use two measures. The first is *word coherence*, $WC^L(V^t)$, which indicates how coherent the L-top words in vector V^t are. The more coherent they are, the better we can represent the set of keywords. We define the word coherence as the average pairwise cosine similarity of the L-top words. Also, in a sense, each L-top words is a cluster containing the relevant words of a term. So we use *Davies-Bouldin Index* [11] as the second measure that optimizes two criteria: (1) minimizing intra-distance between words and the centroid, and (2) maximizing inter-distance between keywords. Values closer to zero indicate a better clustering. The desired value of L should be able to find a sufficient number of similar keywords and provide high word coherence and low Davies-Bouldin index. In our experiments[8], we found that $L = 15$ provides a nice estimation enabling retrieval of at least 10 similar keywords for a given term in 98% cases.

7.3 Experimental evaluation

In this section, we present an extensive set of experiments to evaluate the efficiency and effectiveness of our proposed community search algorithms. Since there is no existing work that considers keywords in influential community search, we first adapt our algorithms to work without keywords and compare the performance with the state-of-the-art influential community search algorithm proposed by

4. https://aminer.org/open-academic-graph
5. http://www.yongliu.org/datasets/index.html
6. https://dl.acm.org/ccs/ccs_flat.cfm
7. https://code.google.com/archive/p/word2vec/
8. https://github.com/saiful1105020/Shared/blob/master/sem_eval.pdf

Li et al. [24]. This experiment demonstrates the superiority of our algorithms in terms of query processing time, and keyword cohesiveness of the retrieved communities. Then, we demonstrate experiments on how to select a good value for system parameter β (to control the trade-off between cohesiveness and influence scores) for different datasets.

The OAG dataset is enriched with metadata from millions of articles, and better fits the application scenarios of our study, and thus we use it as the default dataset. We demonstrate how the query parameters (k_{min}, r) affect the performance of our algorithms. We also show how our algorithms scale with larger datasets. Usually, for the queries with AND predicate, most of the vertices become irrelevant to the query and get filtered out making the query essential graph much smaller than for OR predicates. So, queries with OR predicate are more time consuming and challenging. Therefore, we primarily select OR as the default predicate for the experiments. We show that our algorithms work well for the queries with AND predicate as well. Finally, we present an experiment summarizing the performance of our algorithms in all the datasets keeping the other parameters at the default value.

7.3.1 Comparison with state-of-the-art

We compare our algorithms with existing influential community search studies. We choose Online-All used in Li et al. [24] as the state-of-the-art approach as it can find influential communities in non-attributed graphs. We evaluate the performance of the algorithms in terms of query processing time (efficiency), and keyword cohesiveness (effectiveness).

Efficiency: To compare our algorithms with Online-All, we re-construct the attributed graph as a non-attributed graph by keeping vertices related to a single keyword (randomly determined) and removing the others. We also need to input the cohesiveness parameter k in Online-All. For this, we only consider the top community ($r = 1$), and the value of k in the top community returned by our approach is fed to Online-All. So, Online-All algorithm can return the best k-core community according to the definition of Li et al. [24]. For a fair comparison, we avoid the other algorithms proposed by Li et al. [24] since these algorithms use pre-computed indexes that cannot be adapted for keyword-aware influential community search. Also, we do not compare Online-All with our best approach TREE-EXPLORE, since TREE-EXPLORE also uses a custom pre-computed index. We show the query processing times of these algorithms in Figure 4. Our algorithms are significantly faster compared to the Online-All algorithm, especially for the largest OAG dataset (Figure 4a shows that our algorithms are 4-5 times faster).

Cohesiveness: The effectiveness of a keyword-aware community can be measured in terms of keyword and structural cohesiveness. To compare the keyword cohesiveness, we use the popular community pair-wise Jaccard (CPJ) metric [14]. CPJ measures the similarity of the members of top communities in terms of keywords. We form the queries as described in Section 7.1 to evaluate our approach. Li et al. [24] cannot process queries with keywords; rather, they require a cohesiveness parameter k as input. For the simplicity of presentation, we denote our approach as KICQ-AND, KICQ-OR for AND, OR predicates, respectively. Online-x denotes OnlineAll algorithm in Li et al. [24] with parameter $k = x$. The comparison is presented in Figure 5. For the communities returned by our approach with AND predicate, keyword cohesiveness is 1.3-10 times higher than Li et al., indicating that our approach was more successful in choosing members having similar expertise. Note that our approach and Li et al. [24] use the same community model (i.e., k-core), and the structural cohesiveness is similar.

7.3.2 Selecting β for various datasets

We first run experiments to find an appropriate value for parameter β, which balances the weight of cohesiveness and individual influence (Equation 2). Choosing a good value for β is necessary because, if β is too large, top communities will be highly cohesive but members may have low influence. Again, when β is too small, top communities will contain members with high influence, but the communities may not be cohesive. To find a good choice of β, we consider the network structure, that is: (1) we plot structural cohesiveness measures (i.e., density and average degree [14]), and (2) use the average influence score of the members for different β values as shown in Figure 6.

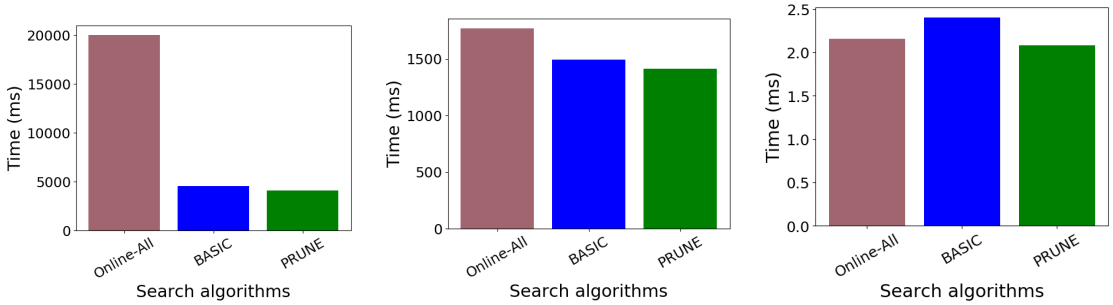
For OAG, communities with higher cohesiveness seem to contain influential individuals. This can be easily explained by the fact that an author who has co-authorship with a large number of authors is likely to have a strong influence in her field of studies. So, for OAG, we choose a high value of β (i.e., 0.6). For Twitter, choosing β near 1 (i.e., 0.95) seems to produce highly cohesive communities, although the average influence is reduced by a small margin. Similarly, we set $\beta = 0.8$ for Gowalla.

7.3.3 Effects of query parameters

The $KICQ$ query takes two additional parameters apart from the keywords and predicates: k_{min} denotes the minimum cohesion factor required for a candidate community, and r denotes the number of top communities to be returned. Here, we present how these parameters affect the efficiency of our algorithms. Also, based on the observations mentioned below, we set $k_{min} = 10$ and $r = 3$ as default while performing other experiments.

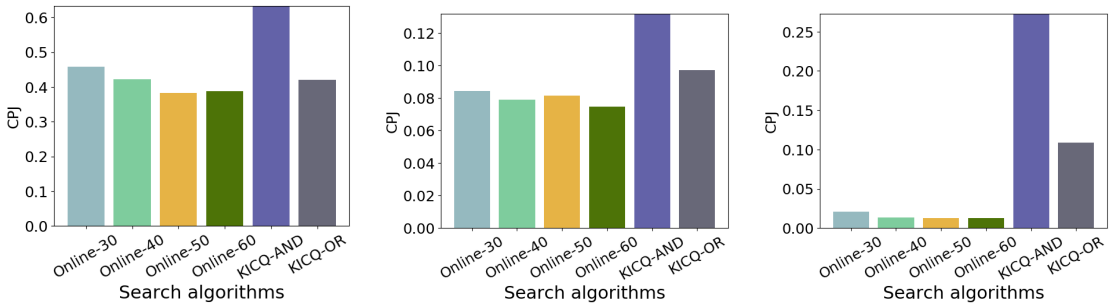
Varying k_{min}: Figure 7 shows how the query processing time is affected by parameter k_{min}. None of the BASIC-EXPLORE and PRUNED-EXPLORE algorithms are significantly affected by the value of k_{min}. However, if k_{min} is set to a high value, TREE-EXPLORE does not need to explore the tree nodes representing k-cores with lower k values. This enables TREE-EXPLORE to skip a larger part of the tree since most of the vertices in the OAG dataset has degree less than 10 making TREE-EXPLORE significantly faster for higher k_{min} values.

Varying r: Figure 8 demonstrates how the query processing time is affected by the number of communities to be retrieved (query parameter r). Both BASIC-EXPLORE and PRUNED-EXPLORE compute core decomposition once on the entire query essential graph. However, TREE-EXPLORE performs the core decomposition on-demand basis. BASIC-EXPLORE is not affected by the



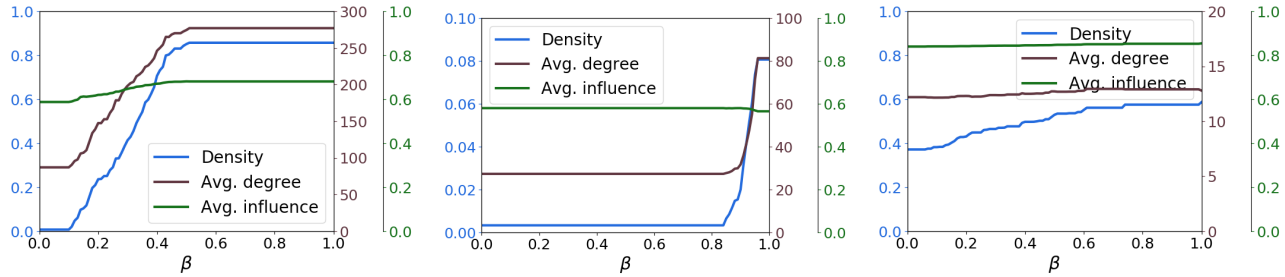
(a) OAG (b) Twitter (c) Gowalla

Fig. 4: Evaluation of efficiency (query processing time)



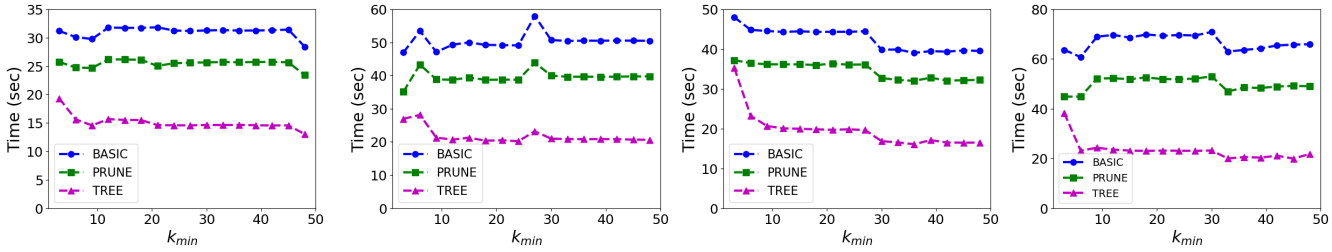
(a) OAG (b) Twitter (c) Gowalla

Fig. 5: Evaluation of effectiveness (keyword cohesiveness)



(a) OAG (b) Twitter (c) Gowalla

Fig. 6: Effectiveness for various values of β



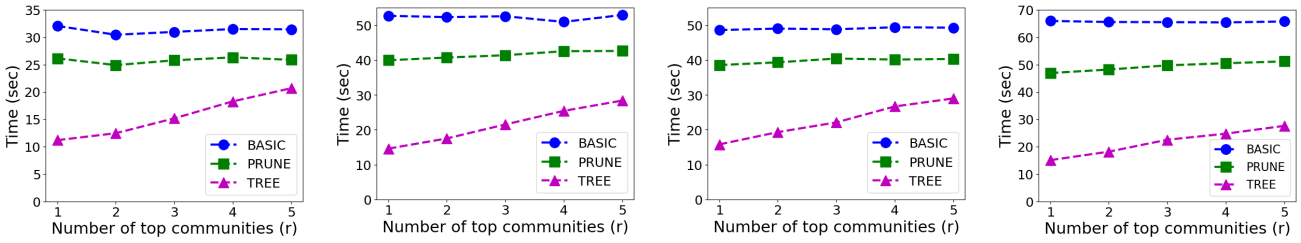
(a) 2 keywords (b) 3 keywords (c) 4 keywords (d) 5 keywords

Fig. 7: Effect of varying k_{min}

value of r since it does not use any pruning based on the retrieved communities. PRUNED-EXPLORE prunes some expansion based on top-r score, but the query processing time is not noticeably affected. The effect is more substantial in TREE-EXPLORE. A significant part of the graph does not require core decomposition if r is small. If r is high, the algorithm can only prune a few tree nodes, but the advantage is ruled out by the overhead of decompressing the graph vertices inside a tree node. However, for small values of r, TREE-EXPLORE significantly outperforms the other algorithms.

7.3.4 *Scalability of KICQ*

Since, most of the social networks are very large, and also grow rapidly, a good search algorithm must be scalable



(a) 2 keywords (b) 3 keywords (c) 4 keywords (d) 5 keywords

Fig. 8: Effect of varying r

(w.r.t. runtime and memory). To show the scalability, we consider different sizes of OAG dataset by varying the number of vertices and thereby edges. We demonstrate how the query processing time and memory requirement (index size) are affected when the network grows in size.

Query processing time: Figure 9 shows that with the increase in the number of vertices, runtime also increases. Clearly, TREE−EXPLORE significantly outperforms the other approaches (1.5-4 times faster than the BASIC−EXPLORE) with varying number of keywords. Also, PRUNED−EXPLORE and TREE−EXPLORE scales much better than BASIC−EXPLORE.

Index size: Figure 10 shows how the size of the graph and corresponding KIC−tree index increase with the increasing number of vertices and keywords. The result conforms to the complexity analysis demonstrated in Section 6.3.2. Index size is linear to both the number of vertices and the number of keywords if one of these remains unchanged. Also, index size is bounded by the graph size.

7.3.5 Performance for AND predicate

We also demonstrate that our approach works well for queries with AND predicate. We compute the query processing time of our algorithms for the various scales of OAG dataset (mentioned in Section 7.3.4). For the queries with AND predicate, the number of relevant vertices is very small compared to the OR predicate. So, both the BASIC−EXPLORE and PRUNE−EXPLORE require to process a very small query essential graph, thereby, perform significantly faster (10-15x faster on average) than the queries with OR predicate. Though the TREE−EXPLORE algorithm has to process the entire pre-computed KIC-tree, which may undermine the advantage of having a small number of relevant vertices, we find that TREE−EXPLORE still outperforms the other two approaches in all cases.

7.3.6 Performance in various datasets

Finally, we present how our algorithms perform in a variety of datasets. For this evaluation, all the parameters are set to their default values. First, we present the query processing time of our algorithms in Table 5 to demonstrate the efficiency. In Gowalla dataset, the number of users for a set of query locations is very small, so the runtime is very small for all the algorithms. It is clear that TREE−EXPLORE algorithm significantly outperforms the other algorithms for OAG and Twitter datasets where each query involves a large number of relevant vertices.

We use popular structural cohesiveness metrics diameter, density, average degree, and clustering coefficient [15] to measure the quality of the communities retrieved by our approach. These measures mostly depend on the community models (e.g., k-core, k-truss) as discussed in a survey of community search [15]. The survey prefers the k-core model because of its high efficiency with minimal sacrifice in structural cohesiveness. By analyzing the cohesiveness measures in Table 6, we claim that our algorithms can retrieve cohesive communities with a small diameter.

Algorithms	Time (ms)		
	OAG	Twitter	Gowalla
BASIC	30477.92	8180.45	2.40
PRUNE	25089.96	7707.77	**2.08**
TREE	**17390.36**	**7344.49**	2.26

TABLE 5: Query processing time for various dataset

Dataset	Density	Average Degree	Clustering Coefficient	Diameter
OAG	0.621	177.897	0.708	2.12
Twitter	0.097	79.343	0.249	3.16
Gowalla	0.579	7.484	0.881	2.70

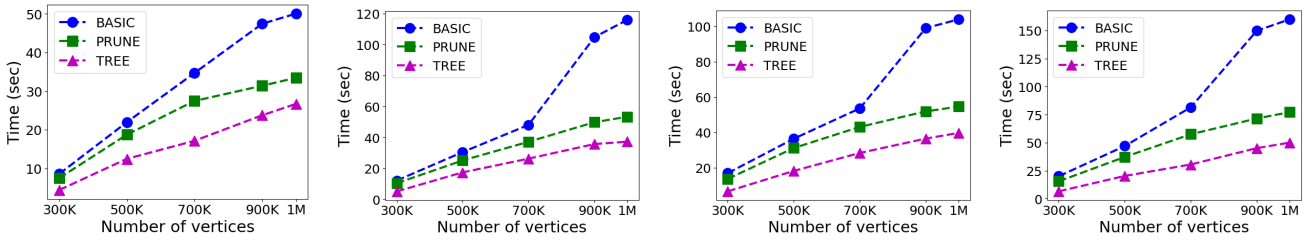
TABLE 6: Structural cohesiveness measures

7.4 A case study

We use a small dataset of a co-author network from ArnetMiner[9] [35] to qualitatively study the effectiveness of our approach. We also observe how our novel influential community measure (Section 5) significantly contributes to enhancing the quality of the retrieved communities. The dataset contains 5,411 vertices and 17,477 edges. Each vertex represents an author annotated with fields from eight different research areas: Data Mining (DM), Web Services (WS), Bayesian Networks (BN), Web Mining (WM), Semantic Web (SW), Machine Learning (ML), Database Systems (DS), and Information Retrieval (IR), where the influence score in each field depends on the number of publications in that field. There is an edge between two authors if they publish at least two papers together.

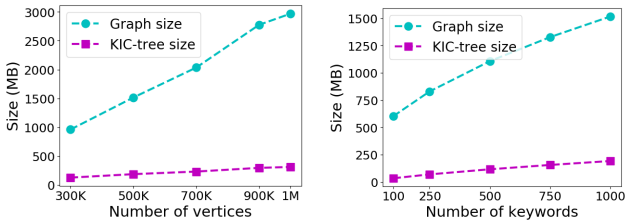
Note that [24] also conducted a case study on this dataset. Figure 11 shows the top community in DS retrieved by our approach (Figure 11a) and by [24] (Figure 11b). The top community returned by our approach is 8-core and thus we compare the result of [24] for $k = 8$. The details, i.e., h-index and the number of citations of each author in our community are shown in Table 7. Among them, the authors who are not included in [24]'s community are shown in bold text. Due to the minimum score modeling, they missed out few top authors in this area including

9. https://aminer.org/lab-datasets/soinf/



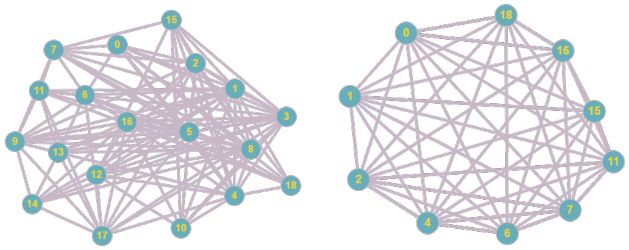
(a) 2 keywords (b) 3 keywords (c) 4 keywords (d) 5 keywords

Fig. 9: Query processing time for varying dataset size



(a) varying vertices (b) varying keywords

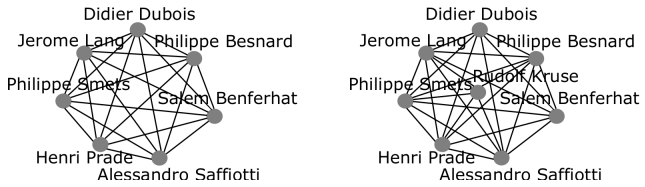
Fig. 10: Index size for varying vertices and keywords



(a) Retrieved by our approach (b) Retrieved by [24].

Fig. 11: Retrieved top communities for DS.

Rakesh Agrawal who was awarded the most influential scholar in the research area of Database Systems (DS) in Aminer [10]. Our approach keeps him in the community as we did not exclude a relatively less influential (with good connectivity) author Laura M. Haas. When Laura is not included in the community, Rakesh Agarwal is connected to less than 8 authors in the community, which turns out to be a non 8-core community. Since in the minimum weight modeling of [24], the inclusion of a low influential member like Laura M. Haas significantly reduces the score of the entire community, the resultant community no longer remains the top community for $k = 8$. These findings show the effectiveness of our problem formulation and score function modeling.



(a) Top most ($k = 6$) (b) Second top most ($k = 5$)

Fig. 12: Top communities for BN OR DM.

10. https://aminer.org/mostinfluentialscholar

Figure 12 presents two top communities for "BN OR DM" returned by our algorithms. The topmost community is fully connected and contains highly influential authors like Didier Dubois (h-index: 125, citations: 82,295), Henri Prade (h-index: 119, citations: 78,700). The second topmost community also contains all the authors from the top-1 community, but the inclusion of another author Rudolf Kruse (h-index: 54, citations: 16,829) increases the contribution of individual scores but decreases the cohesiveness of the community resulting in a lower total score than the first one. This shows the flexibility and trade-off capability among parameters while searching for the communities.

Vertex Id	Author Name	h-index	Citations
0	Hector Garcia-Molina	138	90,220
1	David Maier	65	36,687
2	David J. DeWitt	89	38,770
3	**Philip A. Bernstein**	**80**	**37,823**
4	Michael Stonebraker	72	26,153
5	**Michael J. Franklin**	**34**	**7,746**
6	Serge Abiteboul	80	35,950
7	Jennifer Widom	101	63, 641
8	**Joseph M. Hellerstein**	**90**	**43,207**
9	**Alon Y. Halevy**	**103**	**47,228**
10	**Jim Gray**	**81**	**46,884**
11	Gerhard Weikum	88	34,028
12	**Jeffrey F. Naughton**	**76**	**22,963**
13	**Yannis E. Ioannidis**	**59**	**15,017**
14	**Laura M. Haas**	**49**	**12,834**
15	Stefano Ceri	77	29,506
16	Michael J. Carey	59	16,451
17	**Rakesh Agrawal**	**108**	**124,595**
18	Umeshwar Dayal	62	26,527

TABLE 7: Top communities by our approach in DS.

8 Conclusion

In this paper, we have introduced the keyword-aware influential community query ($KICQ$) that finds top-r most influential communities from an attributed graph. $KICQ$ enables users to find communities of their interests captured as a set of query terms joined by predicates (AND/ OR). We have proposed an influence measure for a community that considers both the cohesiveness and influence of individuals in the community. To efficiently answer the $KICQ$, we have developed a novel KIC-tree index that holds the candidate communities with derived upper bound scores. We propose a baseline solution followed by two efficient algorithms based on the derived upper bounds and results from already explored subgraphs. Our experimental results show that our approach outperforms the state-of-the-art approach in both efficiency (up to 4 times faster) and effectiveness (up to 15 times higher keyword cohesiveness). Finally, we have

validated our proposed influence measure of a community using a case study with a real dataset.

REFERENCES

[1] N. Barbieri, F. Bonchi, E. Galimberti, and F. Gullo. Efficient and effective community search. *Data mining and knowledge discovery*, 29(5):1406–1433, 2015.

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

[3] F. Bi, L. Chang, X. Lin, and W. Zhang. An optimal and progressive approach to online search of top-k influential communities. *PVLDB*, 11(9):1056–1068, 2018.

[4] L. Chen, C. Liu, K. Liao, J. Li, and R. Zhou. Contextual community search over large social networks. In *ICDE*, pages 88–99. IEEE, 2019.

[5] S. Chen, R. Wei, D. Popova, and A. Thomo. Efficient computation of importance based communities in web-scale networks using a single machine. In *CIKM*, pages 1553–1562. ACM, 2016.

[6] Y. Chen, Y. Fang, R. Cheng, Y. Li, X. Chen, and J. Zhang. Exploring communities in large profiled graphs. *IEEE TKDE*, 2018.

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

[8] J. Cheng, Y. Ke, A. W.-C. Fu, J. X. Yu, and L. Zhu. Finding maximal cliques in massive networks. *TODS*, 36(4):21, 2011.

[9] S. Chobe and J. Zhan. Advancing community detection using keyword attribute search. *Journal of Big Data*, 6(1):83, 2019.

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

[11] D. L. Davies and D. W. Bouldin. A cluster separation measure. *IEEE transactions on pattern analysis and machine intelligence*, (2):224–227, 1979.

[12] A. Explosion. spacy-industrial-strength natural language processing in python. *URL: https://spacy. io*, 2017.

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

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

[15] 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*, Jul 2019.

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

[17] 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.

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

[19] Y. Jiang, C. Jia, and J. Yu. An efficient community detection method based on rank centrality. *Physica A: statistical mechanics and its applications*, 392(9):2182–2194, 2013.

[20] A. Khan, L. Golab, M. Kargar, J. Szlichta, and M. Zihayat. Compact group discovery in attributed graphs and social networks. *Information Processing & Management*, page 102054, 2019.

[21] L. Kretz. Virtual online community with geographically targeted advertising, Apr. 17 2008. US Patent App. 11/580,725.

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

[23] R.-H. Li, L. Qin, F. Ye, J. X. Yu, X. Xiao, N. Xiao, and Z. Zheng. Skyline community search in multi-valued networks. In *SIGMOD*, pages 457–472. ACM, 2018.

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

[25] R.-H. Li, J. Su, L. Qin, J. X. Yu, and Q. Dai. Persistent community search in temporal networks. In *ICDE*, pages 797–808. IEEE, 2018.

[26] Y. Liu, A. Niculescu-Mizil, and W. Gryc. Topic-link lda: joint models of topic and author community. In *ICML*, pages 665–672, 2009.

[27] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[28] L. Muflikhah and B. Baharudin. Document clustering using concept space and cosine similarity measurement. In *2009 International Conference on Computer Technology and Development*, volume 1, pages 58–62. IEEE, 2009.

[29] J. Naruchitparames, M. H. Güneş, and S. J. Louis. Friend recommendations in social networks using genetic algorithms and network topology. In *2011 IEEE Congress of Evolutionary Computation (CEC)*, pages 2207–2214. IEEE, 2011.

[30] J. Perkins. *Python 3 text processing with NLTK 3 cookbook*. Packt Publishing Ltd, 2014.

[31] Y. Ruan, D. Fuhry, and S. Parthasarathy. Efficient community detection in large networks using content and links. In *WWW*, pages 1089–1098, 2013.

[32] M. Sachan, D. Contractor, T. A. Faruquie, and L. V. Subramaniam. Using content and interactions for discovering communities in social networks. In *WWW*, pages 331–340. ACM, 2012.

[33] N. Seco, T. Veale, and J. Hayes. An intrinsic information content metric for semantic similarity in wordnet. In *Ecai*, volume 16, page 1089, 2004.

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

[35] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, and Z. Su. Arnetminer: extraction and mining of academic social networks. In *SIGKDD*, pages 990–998, 2008.

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

[37] K. Wang, X. Cao, X. Lin, W. Zhang, and L. Qin. Efficient computing of radius-bounded k-cores. In *ICDE*, pages 233–244. IEEE, 2018.

[38] Y. Wang, L. Wang, Y. Li, D. He, and T.-Y. Liu. A theoretical analysis of ndcg type ranking measures. In *Conference on Learning Theory*, pages 25–54, 2013.

[39] Z. Xu, Y. Ke, Y. Wang, H. Cheng, and J. Cheng. A model-based approach to attributed graph clustering. In *SIGMOD*, pages 505–516. ACM, 2012.

[40] D.-N. Yang, Y.-L. Chen, W.-C. Lee, and M.-S. Chen. On social-temporal group query with acquaintance constraint. *PVLDB*, 4(6):397–408, 2011.

[41] L. Yuan, L. Qin, W. Zhang, L. Chang, and J. Yang. Index-based densest clique percolation community search in networks. *IEEE TKDE*, 30(5):922–935, 2017.

[42] Z. Zhang, X. Huang, J. Xu, B. Choi, and Z. Shang. Keyword-centric community search. In *ICDE*, pages 422–433. IEEE, 2019.

[43] D. Zheng, J. Liu, R.-H. Li, C. Aslay, Y.-C. Chen, and X. Huang. Querying intimate-core groups in weighted graphs. In *2017 IEEE 11th International Conference on Semantic Computing (ICSC)*, pages 156–163. IEEE, 2017.

[44] Y. Zhou, H. Cheng, and J. X. Yu. Graph clustering based on structural/attribute similarities. *PVLDB*, 2(1):718–729, 2009.

[45] Q. Zhu, H. Hu, C. Xu, J. Xu, and W.-C. Lee. Geo-social group queries with minimum acquaintance constraints. *The VLDB Journal*, 26(5):709–727, 2017.

Md. Saiful Islam is a Lecturer at the Department of CSE at Bangladesh University of Engineering and Technology. He is interested in the areas of data mining, natural language processing, social network analysis, and affective computing. He will be joining the PhD program at the University of Rochester in August 2020.

Mohammed Eunus Ali is a Professor in the Department of Computer Science and Engineering (CSE) at Bangladesh University of Engineering and Technology (BUET). He is the group leader of Data Science and Engineering Research Lab (DataLab) at CSE, BUET. His research areas cover a wide range of topics in database systems and information management that include spatial databases, practical machine learning, and social media analytics.

Yong-Bin Kang received his PhD in Information Technology from Monash University in 2011. Currently, he is a data science researcher in the fields of natural language processing, machine learning and data mining at Swinburne University of Technology. He has been working on large industrial, multi-disciplinary research projects such as patent analytics, clinical data analytics, scientific-article analytics, social-media data analytics, expert finding and matching, and machine learning algorithms and applications.

Timos Sellis received the PhD degree in Computer Science from the University of California, Berkeley, in 1986. He is a professor and director of the Data Science Research Institute at Swinburne University of Technology, Australia. Between 2013 and 2015, he was a professor at RMIT University, Australia, and before 2013, the director of the Institute for the Management of Information Systems (IMIS) and a professor at the National Technical University of Athens, Greece. His research interests include big data, data streams, personalization, data integration, and spatio-temporal database systems. He is a fellow of the IEEE and ACM.

Farhana Choudhury received her PhD in computer science from RMIT University in 2017. She is currently a lecturer at The University of Melbourne. Her research interests include spatial databases, data visualization, trajectory queries, and applying machine learning techniques to solve spatial problems.