Keyword-based Socially Tenuous Group Queries

Huaijie Zhu*, Wei Liu*, Jian Yin*, Ningning Cui‡, Jianliang Xu§, Xin Huang§, Wang-Chien Lee¶

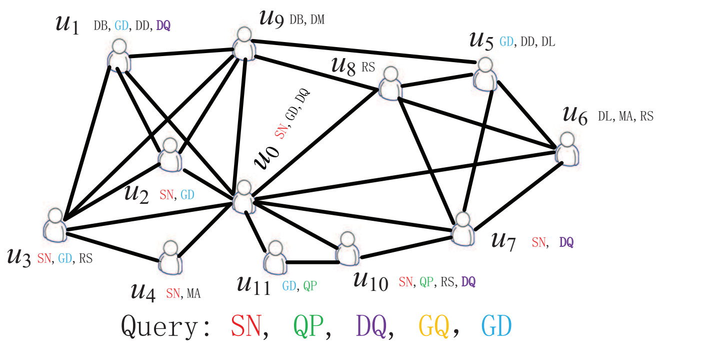
*Sun Yat-sen University, China and Laboratory of Big Data Analysis and Processing, Guangzhou,
‡ An Hui University, §Hong Kong Baptist University, ¶The Pennsylvania State University, USA
{zhuhuaijie, liuw259, issjyin}@mail.sysu.edu.cn, 20210@ahu.edu.cn, {xujl, xinhuang}@comp.hkbu.edu.hk, wlee@cse.psu.edu

***Abstract*—Socially tenuous groups (or simply tenuous groups) in a social network/graph refer to subgraphs with few social interactions and weak relationships among members. However, existing studies on tenuous group queries do not consider the user profiles (keywords) of the members whereas in many social network applications, e.g., finding reviewers for paper selection and recommending seed users in social advertising, keywords also need to be considered. Thus, in this paper, we investigate the problem of *keywords-based socially tenous group* (KTG) queries. A KTG query is to find top N tenuous groups in which the members of each group jointly cover the most number of query keywords. To address the KTG problem, we first propose two exact algorithms, namely *KTG-VKC* and *KTG-VKC-DEG*, which give priority to the *valid keyword coverage* and the combination of *valid keyword coverage and degree*, respectively, to select members to form a feasible group by adopting a branch and bound (BB) strategy. Moreover, we propose *keyword pruning* and *k-line filtering* to accelerate the algorithms. To yield diversified KTG results, we also study the problem of *diversified keywords-based socially tenous group* (DKTG) queries. To deal with the DKTG problem, we propose a *DKTG-Greedy* algorithm by exploiting a greedy heuristic in combination with KTG-VKC-DEG. Furthermore, we design two alternative indexes, namely NL and NLRNL, to efficiently check whether the social distance of any two members is greater than the social constraint k in the above algorithms. We conduct extensive experiments using real datasets to validate our ideas and evaluate the proposed algorithms. Experimental results show that the *NLRNL* index achieves a better performance than the *NL* index.**

***Index Terms*—Tenuous group queries, keyword, social network**

I. Introduction

The problem of finding tenuous groups in social networks, i.e., groups with few social interactions (or weak relationships) among members, has received growing research interests recently [1]–[4]. However, most of the existing works on this problem focus only on the tenuous social relationships among group members without considering their profiles. We observe that in many social network applications, keywords also need to be considered, such as finding reviewers for paper selection and recommending seed users in social advertising. For reviewer selection, it is quite desirable to find reviewers with research expertise to match the topic of a paper under review. For social advertising in marketing campaigns, it is preferable to have seed users not familiar with each other so as to increase the propagation influence. Moreover, the seed users should cover the keywords associated with the product. An example of reviewer selection is shown as follows.



DB: database	MA: machine answering
DD: distributed database	ML: machine learning
DL: deep learning	NLP: natural language processing
DM: data mining	QP: query processing
DQ: data quality	RS: recommending system
GD: graph database	SD: spatial database
GQ: group query	SN: social network

Figure 1. Attributed social networks with keywords.

Example 1: Figure 1 depicts a running example based on an attributed social network. The top part of the figure shows the social relationships among reviewers (vertices) and each reviewer is profiled by a list of keyword terms in the research area of database and data mining. The bottom part lists all the keywords and their abbreviations. Suppose that our goal is to find 3 reviewers whose combined expertise can cover as many keywords of a given paper, e.g., $\{SN, QP, DQ, GQ, GD\}$, as possible. It is very desirable that the reviewers' expertise should match the keywords of the allocated papers in order to ensure proper review and assessment of submitted manuscripts. To ensure the fairness, there should be no close social relationships among all the reviewers to avoid leading unanimous but nonobjective review comments. This is because it may lead to consistent but potentially biased reviews. For example, junior reviewers could be easily influenced by the opinions of senior reviewers. In this figure, u_6 and u_7 should not together be the reviewers because they are directly connected to each other. In addition, the reviewers should cover as many keywords in a paper as possible so as to ensure that the paper is reviewed by experts. For example, u_0 and u_7 do not cover many keywords of the paper, thus it is not ideal to have them assigned together in a reviewer panel. As another example, $\{u_{10}, u_1, u_4\}$ is a good choice because the group members are not closely socially related but do cover many keywords of the paper.

Under the scenarios of reviewer selection and recommending seed users, the need for tenuous group queries that incorporate keywords matching has inspired our research presented in this paper. Specifically, we propose a new tenuous group query, namely, *keyword-based tenuous group* (KTG) query. Consider an attributed social network represented by a graph G. Given a KTG query, specified in the form $\langle W_Q, p, k, N\rangle$, the KTG query aims to find top N groups of size p in which each result group is a tenuous group (i.e., the social distance of any two result members is greater than tenuity constraint k), the members of each result group jointly cover the most query keywords in the query keyword set W_Q (i.e., paper keywords), and each vertex in the result group should contain at least one query keyword. An example of a KTG query is shown in Figure 1. Given a KTG query $\langle W_Q = \{SN, QP, DQ, GQ, GD\}$, p=3, k=1, N=2$\rangle$, the result groups of this KTG query are $\{u_{10}, u_1, u_4\}$ and $\{u_{10}, u_1, u_5\}$, both of which cover the most query keywords $\{SN, QP, DQ, GD\}$.

Dealing with the KTG problem efficiently is challenging due to tenuity and keyword constraints. A simple solution to the KTG problem is the brute-force method, which needs to enumerate all groups of p vertices, and then determine whether they are tenuous groups. If they are tenuous groups, we calculate the number of query keywords which the members jointly cover and return the groups with the highest number of covered query keywords. However, this method is clearly inefficient. To obtain the query result, the method needs to enumerate $C^p_{|V|}$ possible groups, and thus its time complexity is $O(|V|^p)$. Note that, as we show later, the problem of processing KTG is NP-hard. Nevertheless, due to the importance of keyword and social factors considered in KTG, we exploit the keyword and social constraints specified in KTG while forming the candidate groups to alleviate the processing cost. Our idea is to expand the solution space in a systematical way such that we can efficiently find the solution without examining all candidate groups in details. In other words, we incrementally select the members by giving priority to those (a) who maximize the query keywords increment of the result and (b) who are strangers. Finding tenuous groups which best meet both conditions of (a) and (b) is not trivial.

To measure the tenuity of a group, we first introduce a newly designed metric, called *k-distance group*, to formalize the KTG problem. Then we prove that KTG is an NP-hard problem. Fortunately, while the problem is very challenging, it is still tractable since the size of p is relatively small in reviewer selection.

To tackle the KTG problem, we first propose an exact algorithm, namely KTG-VKC, which gives priority to the *valid keyword coverage* to select the users to form a good solution early for KTG queries while exploiting a branch-and-bound method to explore the possible groups. This branch-and-bound exploration idea is equipped with the *keyword pruning* and *k-line filtering* to avoid traversing unnecessary combinations. The valid keyword coverage of a user u is the ratio of the number of query keywords covered by u minus the query keywords already contained in the intermediate result set to the number of query keywords. Moreover, to improve the efficiency, we also propose another new algorithm, namely *KTG-VKC-DEG*, which first gives priority to user's valid keyword coverage and then takes the user's degree to select the users to form the solution. For the vertices in the remaining users set, some vertices have the same valid keyword coverage after the valid keyword coverage sorting. For those vertices with the same valid keyword coverage, we further take priority on sorted vertex degree in descending order. The smaller is the degree of a vertex, the more vertices are unfamiliar with this vertex so that we easily form a good combination earlier. That is, we give priority to the users whose degree is small to form the combinations to find a better combination as early as possible. Moreover, the degree of a vertex does not change as the procedure proceeds, so the computational overhead is small. In addition, for efficiently checking whether the social distance of any two members is greater than the social constraint k, we design two alternative indexes, namely *NL* (h-hop neighbors list) and *NLRNL* ((c-1)-hop neighbors list and reverse c-hop neighbors list).

To support the diversity of the result groups, we extend the KTG query to study the *diversified keyword-based tenuous groups* (DKTG) problem, which maximizes the diversity of the members in the top N result groups. We propose a new algorithm, called *DKTG-Greedy*, by exploiting a greedy heuristic in a combination with KTG-VKC-DEG, to address the DKTG problem. The main idea of *DKTG-Greedy* is to first greedily achieve the largest contribution in the keyword coverage part, and then to maximize the biggest diversity. Finally, we conduct extensive experiments using four real datasets to validate our ideas and evaluate the performance of the proposed algorithms and indexes.

The contributions made in this paper are five-fold:

- We formalize a new variant of tenuous socially group query, *keyword-based tenuous group* (KTG) query, for reviewer selection. To the best of our knowledge, this is the first attempt to tackle the KTG problem.
- We propose two exact algorithms, namely KTG-VKC and KTG-VKC-DEG, which give priority to the *valid keyword coverage* and the combination of *valid keyword coverage and degree*, respectively, to select users to form a feasible group while exploiting a branch and bound (BB) strategy, equipped with *keyword pruning* and *k-line filtering*.
- We also design two alternative indexes, namely *NL*, and *NLRNL*, for efficiently checking whether the social distance of any two members is greater than k.
- In addition, we formalize and study the diversified keyword-based tenuous groups (DKTG) problem, and propose the *DKTG-Greedy* algorithm to exploit a greedy heuristic combination with the above KTG-VKC-DEG approach, to address the DKTG problem.
- We conduct extensive experiments to evaluate the proposed algorithms and indexes. Experimental results show that KTG-VKC-DEG significantly outperforms the KTG-VKC algorithm.

The rest of this paper is organized as follows. Section II reviews the related work. Section III gives some basic definitions and formulates the KTG problem. Section IV presents our two exact algorithms in detail. Section V shows how to check whether the social distance of any two members is greater than the social constraint k, by using *NL* and *NLRNL* indexes. Section VI extends this work to study the DKTG problem and proposes a new algorithm upon KTG-VKC-DEG. Section VII reports the experimental results and our findings. Finally, Section VIII concludes the paper.

II. Related Work

In this section, we review two lines of research efforts, *tenuous group queries*, and *diversification of query results*, which are relevant to our work in the aspects of social group queries and keyword search.

A. Tenuous Group Queries in Social Networks

Different from dense group query/search, which is to find a dense group/community in social network, e.g., k-core [5]–[9], k-truss [10]–[12], cliques and motifs [13], [14], the tenuous group queries are proposed to find a group/subgraph with few social interactions (or weak relationships) among members. Several metrics are proposed to measure a socially tenuous group, such as density [15], [16], the number of edges [17], k-triangle [1], [4], k-line [2], k-tenuity [3], and so on. First, the density and the number of edges do not guarantee that there exist some directly connected edges in a tenuous group. Simply limiting the number of edges to zero can overcome this shortcoming, but it does not show the tenuity value of different vertices pairs when there is no directly connected edge. k-line is a concept proposed in the literature [2]. For vertices u and v, (u, v) is a k-line if the hop count of the shortest path between u, v is less than or equal to a given parameter k. When the group has more k-lines, the tenuity of the group is worse. For a triple (u, v, w), the triple is a k-triangle if the hop count of the shortest path between every two nodes in it is less than k. The smaller the number of k-triangles, the higher the tenuity of the group. However, Li [2] finds that k-triangles become invalid when there is no k-triangles but only k-lines in the graph. Thus, [2] proposes the definition of k-line, which allows the existence of k-line in a group, just to ensure that the number of k-line is as small as possible. The idea of [2] is to minimize the number of k-lines in a subgroup, while our problem returns the tenuous groups that do not have any k-line. Moreover, our goal is to jointly maximize the keywords coverage of groups instead of minimizing the number of k-lines. Although Li et al. [18] investigate the tenuous group query by taking the keywords into consideration, our work is different from it in three aspects. First, [18] is to maximize the average coverage of query keywords for the result group. Obviously, this may select some reviewers whose keywords do not contain any of the query keywords. Instead, we require the group to jointly cover as many query keywords as possible and each reviewer should cover at least of the query keywords. Second, Li et al. [18] propose the definition to measure the tenuity of a group, namely k-tenuity, which is the ratio of the number of node pairs within k hops to the total number of node pairs in a group. Obviously, according to the definition of k-tenuity, as long as the value of k-tenuity is greater than 0, there must be at least one node pair within k hops. In a extreme case, there may be neighbors (i.e., 1-hop) in the returned group. Thus, to some extent, this method can not guarantee the tenuity of a group and is different from our work. Third, we consider the diversity of group results to enrich the results.

B. Diversification of the Query Results

The diversification of query results has attracted a lot of attention recently as a method to improve the quality of results by balancing similarity (relevance) to a query q and dissimilarity among results [19]–[23]. In the literature, some specific problems have been studied, such as top-r query [24], [25], skyline query [26], keyword search [27] [39], document retrieval [28], [29], pattern matching [30], maximal clique search [31], coherent core [32] and (k, r)-core [33] by taking diversify into consideration. Diversification has also been considered in keyword search over graphs and databases, where the result is usually a subgraph that contains the set of query keywords. In conventional (nondiversified) keyword search methods, a set of results usually consists of many duplicated answers that contain the same set of vertices (i.e., vertices containing a query keyword). Thus, it's desirable to avoid many similar answers with minor differences [34]. Specially, PerK [35], DivQ [27], and Cai et al. [36], address this problem by using Jaccard distance on the set of nodes of the results, i.e., by considering the common nodes. Similarly, we also utilize the Jaccard distance on the result groups in our work.

However, the techniques in the above-mentioned works can not be applied to solve our proposed diversified KTG queries To the best of our knowledge, our study is the first attempt to incorporate diversified top-N search in tenuous group queries.

III. Problem Formalization

The attributed social network G can be represented as a triple $G = (V, E, \kappa)$, where $V = \{v_i | 1 \leq i \leq n\}$ is the set of vertices, n is the number of vertices, in G, $E = \{v_i, v_j | 1 \leq i, j \leq n\}$ is the set of edges, representing the relationship between two vertices in the network, $\kappa = \{k_1, k_2, \cdots, k_m\}$ represents a set of keywords in the network, and m is the number of all keywords. Each vertex v is associated with a set of keywords $k_v \subseteq \kappa$.

Definition 1: **Social distance.** The social distance of a vertex pair $\{u, v\}$ in G, denoted as $Dis(u, v)$, is the hop of shortest path from u to v.

In this work, our goal is to find a set of vertices that are tenuous (i.e., unfamiliar with each other). To measure the tenuity of a group, we first introduce a metric, called *k-line*, previously proposed by [2].

Definition 2: **k-line.** A k-line is a vertex pair $\{u, v\} \in G$ where $Dis(u, v) \leq k$.

The definition of k-line only measures the tenuity between two vertices, instead of the tenuity of a whole group. Therefore, we give the definition of the tenuity of a whole group based on k-line. In this paper, the notion of *k-distance group* is proposed to guarantee the tenuity between any two vertices in a group.

Definition 3: **k-distance group.** Given an attributed social network $G = (V, E, \kappa)$, a k-distance group F is a subgraph of G where the social distance between any two vertices in F is greater than k, i.e., $\forall u, v \in V_F, Dis(u, v) > k$.

In the above, V_F is the vertices set in F. Group F is a k-distance group, indicating that there are no interrelated pairs of vertices with distances less than or equal to k, and therefore F is tenuous.

Accordingly, we define the tenuity of a group below.

Definition 4: **The tenuity of a group.** Given an attributed social network $G = (V, E, \kappa)$, the tenuity of a group F is the smallest social distance between any two vertices in F.

Property 1: **(Increase.)** When $k_1 > k_2$, the tenuity of the k_1-distance group is greater than the tenuity of the k_2- distance group, and the k_1- distance group must be the k_2- distance group.

Property 2: **(Inclusion.)** Suppose that $F_1 \subset F_2$, if F_2 is a k-distance group, then F_1 is must be a k-distance group.

In addition to ensuring the result group to be a k-distance group, we also need to ensure that the group covers the maximum number of query keywords. We define the query keyword coverage for a vertex below.

Definition 5: **Query keyword coverage of a vertex.** Given a vertex v with keywords set k_v, a query keyword set W_Q, the query keyword coverage of a vertex v is the ratio of the query keywords associated with v to the total number of query keywords, i.e., $QKC(v) = \frac{|k_v \cap W_Q|}{|W_Q|}$.

Based on query keyword coverage of a vertex, we further define the query keyword coverage of a group, as shown in Definition 6.

Definition 6: **Query keyword coverage of a group.** Given a query keyword set W_Q, the query keyword coverage of a group F is the ratio of the query keywords associated with vertices in V_F to the number of query keywords, i.e., $QKC(F) = \frac{|\bigcup_{v \in V_F}(k_v \cap W_Q)|}{|W_Q|}$.

According to Definition 6, a higher query keyword coverage of a group F indicates that it covers more query keywords, and thus group F has more potential as one of the result groups.

As shown in Figure. 1, assume $F_1 = \{u_5, u_7\}$, $F_2 = \{u_4, u_6\}$, and $W_Q = \{SN, QP, DQ, GQ, GD\}$. According to Definition 5, the the query keyword coverage for the vertices u_4 and u_6 in group F_2 is 0.2 and 0.4, respectively. According to Definition 6, the query keyword coverage for F_1 and $F2$ is 0.2 and 0.6, respectively. Thus, F_2 is more suitable as a result than F_1.

We formally define the KTG problem as follows.

Definition 7: **KTG.** Given an attributed social network $G = (V, E, \kappa)$, a query keyword set W_Q, a tenuity constraint parameter k, an integer N, and a tenuous group size parameter p, a KTG query finds the top N groups set RG such that each result group $g \in RG, QKC(g) \geq QKC(g')$ where $g' \notin RG$, where each group in RG is a k-distance group of size p, and each user in the result group should cover at least one query keyword. Each result group g should satisfy the following condition.

- $\forall Dis(u, v)_{(u,v \in V_g)} > k$.
- $|g| = p$.
- For each $v, 0 < QKC(v) \leq 1$.

Theorem 1: The KTG query is NP-hard.

Proof: We prove the NP-hardness of the KTG problem by reduction from a decision version of the well-known Independent Set of size p problem, i.e., checking whether there exists a non-empty independent set of size n in graph $G(V, E)$. We construct an instance of KTG for graph $G(V, E)$ with query $\langle W_Q, p, k, N\rangle$, by setting the parameters $n = p$, k=1, N=1 and $W_Q = \kappa$ such that every user $v \in V$ satisfies $\forall QKC(v) > 0$. k=1 means that there is no edge connecting any two vertices, In other words, the KTG problem is to find an independent set of size n. Thus, the decision problem of independent set of size n is a Yes-instance *if and only if* the corresponding case of our KTG is also a Yes-instance. This completes the proof.

A naive idea regarding how to tackle the KTG problem is the brute force method. In detail, we first enumerate all the possible combinations/groups of p vertices, followed by determining whether each group is a k-distance group. If it is, we then calculate its query keyword coverage to select the N groups with the highest coverage. However, this method is clearly inefficient. To explore all the possible result groups, the method needs to enumerate $C^p_{|V|}$ possible combinations, and thus its time complexity is $O(|V|^p)$.

IV. Exact Algorithms

To address the KTG problem, we first propose two exact algorithms.

A. KTG-VKC

As mentioned, a naive idea to solve the problem is to enumerate every possible combination of p members and choose the top N groups with the highest keyword coverage. Thus, we adopt a branch and bound strategy to explore the possible combinations.

Specifically, let S_I and S_R denote the intermediate result set and the remaining set of candidate users, respectively. At first, S_I is empty and S_R is V. In each subsequent iteration, we select a vertex from S_R to S_I. S_I is considered as a feasible solution when it contains p vertices and follows the constraints of the KTG problem. Then, to optimize the feasible solution, our algorithm backtracks to the previous iteration and the previous S_I and selects another vertex from S_R to S_I while maintaining a branch-and-bound tree to keep track of the backtracking exploration process.

To accelerate the exploration process, it is essential to design some pruning/filtering strategies to avoid the exploration of the invalid combinations. To carry out effective pruning or filtering, obtaining a feasible group with high keyword coverage early is desirable, since the goal is to maximize the keyword coverage of the result group. Obtaining a feasible solution with high keyword coverage early is highly related to selecting which vertex first from S_R to S_I. Thus, the KTG-VKC algorithm gives priority to the users whose valid keyword coverage to query keyword is high (also referred to as *valid keyword coverage sorting.*) when performing the branch and bound exploration.

The valid keyword coverage of a vertex is defined as follows.

Definition 8: **Valid keyword coverage of a vertex w.r.t. an intermediate result set.** Given a vertex v, an intermediate result set S_I and a query keyword set W_Q, the valid keyword coverage of a vertex to query keyword set W_Q is the ratio of the query keywords covered by vertex v divided by the query keywords already covered by S_I to the number of query keywords, i.e., $VKC(v) = \frac{|k_v \cap W_Q - \bigcup_{u\in S_I}(k_u \cap W_Q)|}{|W_Q|}$.

An example of valid keyword coverage sorting is illustrated in Figure 1. Suppose $S_I = \{u_0\}$, $S_R = \{u_1, u_2, u_5, u_{10}\}$, $W_Q = \{SN, QP, DQ, GQ, GD\}$. For the valid keyword coverage sorting, we first compute the valid keyword coverage for each vertex in S_R and sort them according to the corresponding value in a descending order. The valid keyword set covered by u_0 is $\bigcup_{u\in S_I}(k_u \cap W_Q) = \{SN, GD, DQ\}$. We then compute the set of valid keywords for each vertex in S_R, $k_{u_1} \cap W_Q = \emptyset$, $k_{u_2} \cap W_Q = \emptyset$, $k_{u_5} \cap W_Q = \emptyset$, and $k_{u_{10}} \cap W_Q = \{QP\}$. Accordingly, we obtain the valid keyword coverage for each vertex in S_R, $VKC(u_1) = 0$, $VKC(u_2) = 0$, $VKC(u_5) = 0$, $VKC(u_{10}) = 1/2$. Sorted by $VKC(u)$ in the descending order, S_R becomes $\{u_{10}, u_1, u_2, u_5\}$.

Note that the valid keyword coverage is different from the query keyword coverage (as shown in Definition 3). Therefore, we can also exploit the query keyword coverage to sort the vertices. The advantage of this sorting is that we only need to calculate query keyword coverage once for each vertex and only need sorting once, while for valid keyword coverage sorting, we should compute it for different S_I and require continuous sorting. However, the query keyword coverage sorting does not make S_I always have the largest increment of keyword coverage, thus it can not produce the group with high keyword coverage and does not facilitate the pruning. For comparison, we evaluate both sorting strategies in Section VII.

Keyword pruning. For pruning the invalid branches, we propose a keyword pruning strategy (as shown in Theorem 2), which effectively filters those vertices in S_R not qualified to be a member of a better result group by calculating upper bounds on the number of valid keywords for S_I and the number of valid keywords for S_R.

Theorem 2: Given the N-th group H in the currently best top N groups and the current intermediate group S_I, S_I can not form a qualified result group iff

$$QKC(S_I) + \sum_{(v\in top\ p-|S_I|\ in\ S_R)} VKC(v) \le QKC(H). \tag{1}$$

For S_R, here we only visit vertices with the valid keyword coverage values in the top $p - |S_I|$, and $\sum_{(v\in top\ p-|S_I|\ in\ S_R)} VKC(v)$ is the sum of the valid keyword coverage of the top $p - |S_I|$ vertices. Since the valid keyword coverage of the top vertices can be obtained from the valid keyword coverage ranking, keyword pruning is not time-consuming.

Consider an example with p=2 and $W_Q = \{SN, QP, DQ, GQ, GD\}$, as shown in Figure 1. Assume that the current N-th group g= $\{u_{10}, u_1\}$, and we update its valid keyword coverage value as $\frac{4}{5}$. When exploring the branch $S_I = \{u_4\}$ and $S_R = \{u_5, u_6, u_9\}$, since $\frac{|\bigcup_{u\in S_I}(A_u \cap W_Q)|}{|W_Q|} + \sum_{(v\in top\ p-|S_I|\ in\ S_R)} VKC(v) = \frac{1}{5} + \frac{1}{5} = \frac{2}{5} <$ QKC(g)=$\frac{4}{5}$, we safely prune the branch $S_I = \{u_0\}$, and backtrack to its previous state according to the keyword pruning.

With regard to $VKC(v) = \frac{|A_v \cap W_Q - \bigcup_{u\in S_I}(A_u \cap W_Q)|}{|W_Q|}$, we can see that the valid keyword coverage of the vertex is related to S_I. When S_I changes, we need to update the valid keyword coverage of the vertices in S_R.

Note that the algorithm equipped with the keyword pruning only removes the unqualified groups with low query keyword coverage. To further filter the invalid groups, we propose a *k-line filtering* strategy, which removes the infeasible groups (groups that do not satisfy the k-distance group).

Theorem 3: Given the current intermediate set S_I and the remaining candidate vertex set S_R, for a vertex $v \in S_R$, v can be safely filtered iff there exists one vertex $u \in S_I$ so that v, u is a k-line.

k-line filtering. At each iteration, after selecting a vertex from S_R to S_I, k-line filtering removes vertices in S_R that form a k-line group with that vertex. Obviously, we do not need to consider those deleted vertices. If they are selected into S_I, S_I is not a k-distance group. Then the result obtained from expanding S_I is not a k-distance group (the inverse proposition of Prop. 2), which is not consistent with the definition of the KTG problem, thus these vertices should be filtered from S_R.

For example, as in Figure 1, assume that $S_I = \{u_8\}$, $k = 2$. The vertices that can form a k-line with S_I are u_0, u_7, u_3, u_4, u_6. When any of these vertices join S_I, S_I will not be a k-distance group (i.e., it does not satisfy the constraints of the KTG problem), so S_R becomes $\{u_1, u_2, u_5, u_9\}$ after the k-line filtering.

The pseudo-code of KTG-VKC is shown in Algorithm 1. Initially, we remove the unqualified users whose keywords do not contain at least one query keyword and S_R is initialized with the remaining qualified users. At each iteration, we first check whether the size of S_I is equal to p. If it is, we check whether the query keyword coverage of S_I is greater than C_{max} and update the result if possible (Lines 2-4). If the size of S_I is smaller than p, we fist check whether S_I can be pruned through the keyword pruning. If it is not pruned, we put the first element in S_R (e.g., $S_R[0]$) into S_I and delete it from S_I (Lines 8-10). Then we invoke the k-line filtering to filter those unqualified users from S_R and obtain the valid keyword coverage of each users with respect to the new S_I (i.e., $tempS_I$) (Lines 11-14). These remaining qualified users in S_R are sorted again using $SortSR$ according to the new valid keyword coverage of each user. After that, the KTG-VKC algorithm is invoked recursively with $tempS_I$ and $SortSR$ to explore the possible groups (Lines 15).

Algorithm 1: The KTG-VKC Algorithm

```
Input: A KTG query $\langle W_Q, p, k, N\rangle$, an intermediate
       solution set $S_I$, a remaining qualified users set
       $S_R$, and the current highest keyword coverage
       $C_{max}$
Output: Top $N$ KTG result groups $RG$
1  if $|S_I| == p$ then
2      if $QKC(S_I) > C_{max}$ then
3          updateRS($R$, $C_{max}$, $S_I$);
4      return;
5  else
6      if
       $QKC(S_I) + \sum_{(v \in top\ p-|S_I|\ in\ S_R)} VKC(v) \leq C_{max}$
       then
7          return; /* keyword pruning; */
8      $tempS_I \leftarrow S_I$;
9      $tempS_I$.push_back($S_R[0]$);
10     delete $S_R[0]$ from $S_R$;
11     for each user $u_i \in S_R$ do
12         if ! kline_filtering($u_i, tempS_I$) then
13             $VCR(u_i)$= getVKC($u_i, tempS_I, W_Q$);
14             insert $u_i$ into sorting users set $SortSR$ by
               $VCR$ ;
15     KTG-VKC($tempS_I$, $SortSR$);
16 return $RG$;
```

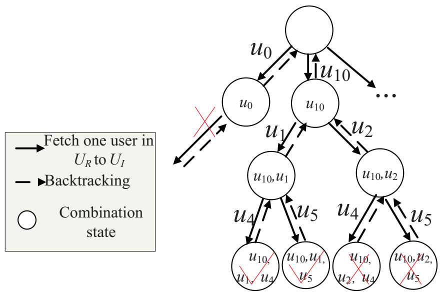


Figure 2. An exmample of KTG-VKC.

An example of KTG-VKC is shown in Figure 2. Given a KTG $\langle W_Q = \{SN, QP, DQ, GQ, GD\}, p = 3, k = 1, N = 2\rangle$ query, at first, we remove the unqualified users and the remaining users set S_R is $\{u_0, u_1, u_2, u_3, u_4, u_5, u_7, u_{10}, u_{11}\}$. According to the valid keyword sorting in the descending order, S_R becomes $\{u_0, u_{10}, u_1, u_2, u_3, u_7, u_{11}, u_4, u_5\}$. Then we invoke the KTG-VKC algorithm to explore the candidate groups as shown in Figure 2. Firstly, S_I is initialized with $\{u_0\}$ according to the sorting. At the same time, S_R is sorted again according to the valid keyword coverage with k-line filtering, and S_R becomes $\{u_5\}$. At this stage, the size of S_R is smaller than 3-1=2, thus branch $\{u_0\}$ is finished. After that, we continue to select u_{10} into S_I and form a new branch $\{u_{10}\}$ and $S_R = \{u_1, u_2, u_3, u_4, u_5\}$. Then, u_1 is selected from S_R and S_I becomes $\{u_{10}, u_1\}$. At this time, S_R becomes $\{u_4, u_5\}$. By selecting a vertex from S_R to S_I iteratively, we first get two result groups $\{u_{10}, u_1, u_4\}$, and $\{u_{10}, u_1, u_5\}$ of size 3 with the query keyword coverage 0.8, sequentially. Branch $\{u_{10}, u_1\}$ is finished and KTG-VKC backtraces to the branch $\{u_{10}\}$ and u_2 is selected to form $\{u_{10}, u_2\}$. Similarly, two groups $\{u_{10}, u_2, u_4\}$, and $\{u_{10}, u_2, u_5\}$ are obtained. However, the query keyword coverage of these two groups are not greater than 0.8, thus they cannot update the result groups. Continuing with the above steps, since there is no better result groups whose query keyword coverage is greater than 0.8, the query processing ends. The final top-2 result groups are $\{u_{10}, u_1, u_4\}$ and $\{u_{10}, u_1, u_5\}$.

KTG-VKC, combining the valid keyword coverage while performing the branch and bound exploration, serves as a natural baseline in our problem. However, this idea has some pitfalls, i.e., the social distance of the selected users is greater than k and KTG-VKC is not efficient in forming a good solution early for pruning and filtering. Thus, we propose a new algorithm, namely *KTG-VKC-DEG* in the following.

B. KTG-VKC-DEG

The valid keyword coverage sorting maximizes the increment of valid keyword coverage of S_I, which allows us to get a feasible group of size p with a high valid keyword coverage early. However, some vertices in S_R have the same valid keyword coverage after the valid keyword coverage sorting. For those vertices with the same valid keyword coverage, we utilize the degree sorting (i.e., sorting by the vertex degree in descending order), as a tie-breaker. That is, the smaller the vertex degree is, the higher priority we choose this vertex in from S_R into S_I since this vertex is less likely be familiar with other vertices. Therefore, in this section we combine the above proposed valid keyword coverage sorting with the degree sorting to propose a new algorithm, called *KTG-VKC-DEG*, which improves the query efficiency of the KTG problem.

Notice that in KTG-VKC-DEG, the degree of vertex does not change as the query processing proceeds. Thus, it only needs to be calculated once and doesnt take a great deal of query time.

With regard to the combination of valid keyword coverage sorting with the degree sorting to rank the users in S_R, we first use the valid keyword coverage to sort the vertices, and for vertices with the same valid keyword coverage, we then sort the vertices according to their degrees. According to this combination, selecting vertices from S_R to S_I at each iteration can guarantee the size of S_I on the basis of ensuring the maximum increment of effective keyword coverage of candidate vertices in S_R, forming a feasible solution as early as possible and improving the pruning efficiency.

Example 2: Recall the example in Figure 1 for illustration of the KTG-VKC-DEG algorithm. Give a KTG $\langle$ $W_Q = \{SN, QP, DQ, GQ, GD\}$, p=3, k=1, N=2$\rangle$ query. Similar to KTG-VKC, we first fetch the remaining users set S_R= $\{u_0$, u_1, u_2, u_3, u_4, u_5, u_7, u_{10}, $u_{11}\}$ be removing those unqualified users. Different from KTG-VKC, KTG-VKC-DEG utilizes the valid keyword sorting and degree as the sorting, S_R becomes $\{u_{10}$, u_0, u_{11}, u_1, u_2, u_3, u_7, u_5, u_4 $\}$. Then we invoke the KTG-VKC-DEG to explore the candidate groups, similar to the process in Figure 2. Firstly, S_I is initialized with $\{u_{10}\}$. At the same time, S_R is sorted again according to the valid keyword coverage and degree, and becomes $\{u_5$, u_1, u_2, u_3, u_4 $\}$. Then u_5 is selected to form a new branch $\{u_{10}, u_5\}$ and $S_R = \{$ u_1, u_2, u_3, u_4 $\}$. By selecting a vertex from S_R to S_I iteratively, we first get two result groups $\{u_{10}, u_5, u_1\}$, and $\{u_{10}, u_5, u_2\}$ of size 3 with the query keyword coverage 0.8, sequentially. Notice that the feasible groups $\{u_{10}, u_5, u_3\}$ and $\{u_{10}, u_5, u_4\}$ have the same query keyword coverage, they can not update the current result groups. After that, we move forward on the branch S_I= $\{u_0\}$, and U_R becomes $\{u_5$, $u_7\}$. Then we select u_5 from U_R and S_I= $\{u_0, u_5\}$. At this time, S_R becomes empty since u_7 is filtered due to k-line filtering. Similar to the above steps, branches $\{u_{11}\}$, $\{u_1\}$, ..., $\{u_7\}$ are explored. Since there is no better result groups that have query keyword coverage greater than 0.8, the query processing ends and the final top-2 result groups are $\{u_{10}, u_5, u_1\}$, and $\{u_{10}, u_5, u_2\}$. Compared with the two examples of KTG-VKC and KTG-VKC-DEG, KTG-VKC-DEG gets the result groups with high query keyword coverage earlier than KTG-VKC since KTG-VKC requires to explore the branch $\{u_0\}$ first and then the branch $\{u_{10}\}$.

Discussion. To handle the scenarios in which the authors are familiar with the reviewers, our techniques can be extended to handle the query including multiple query vertices (i.e., the authors). The main idea is to remove those reviewers who are familiar with the authors, i.e., only reviewers whose social distance from the authors is greater than k remain.

V. Index-based Algorithm for Fast Social Distance checking

Notice that in k-line filtering, checking whether the social distance of two members is greater than k is an important operation. Moreover, in our KTG-VKC algorithm, this operation is performed many times. Accordingly, in this section, we propose an efficient algorithm to check the social distance of two members is greater than k. Inspired by the 1-hop or 2-hop label index [37], we first propose an h-hop neighbors list index, called *NL* index.

A. Using h-hop Neighbors List (NL) Index

To design a good h-hop neighbor index, we consider the value of tenuity constraint k and the number of h-hop neighbors of each vertex.

Checking whether the social distance of two members u_i, u_j is greater than k can be processed in two aspects, (1) checking whether u_i is one of the u_j' l-hop neighbors where l=1, 2, ..., k. If it is, their social distance is not greater than k; Otherwise, their social distance is greater than k. Or (2) checking whether u_i is one of the u-hop neighbors where u= k+1, k+2, ..., k_{max}. If it is, their social distance is greater than k.

Thus, for maintaining the top h-hop neighbors list, there exist two cases. Case 1: h is greater than k. We just check whether member u_i is one of the u_j' 1-hop, 2-hop,..., k-hop neighbors. Case 2: h is not greater than k. We first should check whether u_i is one of the u_j' 1-hop, 2-hop,..., h-hop neighbors. If it is not, we continue to check whether u_i is one of the u_j' (h+1)-hop, (h+2)-hop,..., k-hop neighbors. For this situation, we expand the neighbors of h-hop neighbors as (h+1)-hop neighbors, the neighbors of (h+1)-hop neighbors as (h+2)-hop neighbors until the k-hop neighbors or u_i is one (h+i)- hop neighbor. Moreover, with regard to expanding neighbors, we remove the neighbors who are already one of m-hop neighbors where $m < h$+1 when obtaining (h+1)-hop neighbors. The pseudo-code of checking the social distance using the top h-hop neighbors list is shown in Algorithm 2.

Algorithm 2: CheckSocialDistance using NL Algorithm

```
Input: Users $u_i$ and $u_j$, top $h$-hop neighbors list $L$, $k$
Output: Greater than $k$ or not
1  if $h > k$ then
2      for i=1,2,...,$k$ do
3          /* $L[u_j][i]$ denotes the i-hop neighbors of $u_j$; */
4          if $u_i$ is in $L[u_j][i]$ then
5              return false;
6  else
7      for i=1,2,..., $h$ do
8          if $u_i$ is in $L[u_j][i]$ then
9              return false;
10     for $j$=$h$+1, $h$+2,..., $k$ do
11         $L[u_j][j+1]$ = expandNeighbor($L[u_j][j]$);
12         if $u_i$ is in $L[u_j][j+1]$ then
13             return false;
14 return true;
```

Since the cost of expanding the neighbors of h-hop neighbors to be (h+1)-hop neighbors depends on the number of h-hop neighbors and the number of corresponding neighbors, we choose the number of m-hop neighbors with the maximal one as h value. If h is not greater than k, this h-hop neighbors may facilitate checking the social distance and thus reduce the cost of expanding neighbors. However, this may incur some space overhead to store the maximum number hop neighbors.

We use an example to illustrate how to check the social distance using h-hop neighbors. Assume that we need to check whether the social distance of u_3 and u_5 is greater than 3. We have maintained the top 2-hop neighbors of u_3 (e.g., its 1-hop neighbors are u_0, u_2, u_4, u_9 and 2-hop neighbors are u_6, u_7, u_8, u_{10}, u_{11}). Based on Algorithm 2, we first compare h=2 with k=3. Since $h < k$, we first check whether u_5 is one of 1-

hop neighbors and 2-hop neighbors of u_3. Since u_5 is not one such neighbor, we require to expand 2-hop neighbors to 3-hop neighbors and check whether u_5 is one of 3-hop neighbors. We find that u_5 is one of the 3-hop neighbors of u_3 and ascertain that the social distance of u_3 and u_5 is not greater than 3.

If h is not greater than k, we still need to expand h-hop neighbor to (h+1)-hop neighbors, (h+2)-hop neighbors and even k-hop neighbors in the worst case. The expanding cost may be high. Due to this worst case scenario, we need to check whether one user u_i is one of the u-hop neighbors where u= k+1, k+2,..., k_{max}, we develop another new index.

B. Using NLRNL Index

To avoid expanding the h-hop neighbor to (h+1)-hop neighbors, we propose another index, made up of (c-1)-hop neighbors list and reverse c-hop neighbors list, called *NLRNL* index. In contrast to (c-1)-hop neighbor, the reverse c-hop neighbors index maintains the neighbors whose social distance is greater than c. For quickly checking whether the social distance of two users is greater than k, choosing an appropriate c value is important. Notice that in NLRNL index, we do not store c-hop neighbors. Thus, we hope the number of c-hop neighbors is the largest one among the 2-hop neighbors, 3-hop neighbors,..., and so on. Since the number of c-hop neighbors is different for each vertex, we keep various c for each vertex. To obtain such a c value, our idea is to compute the number of the 2-hop neighbors, 3-hop neighbors, and so on, and choose the neighbors with the maximum number as c for each vertex.

Once we have built the NLNRL index, we utilize it to check the social distance of two members as follows. We first check whether c-1 is not less than k. If it is, we check u_i is one of the i-hop neighbors of u_j where i= 1, 2,..., k. If it is one such neighbor, we can determine the social distance of u_j and u_i is not greater than k. Otherwise, the social distance of u_j and u_i is greater than k. If c-1 is greater than k, we just check the corresponding reverse neighbors list. If u_i is one of reverse j-hop neighbors where j=k+1, k+2, k+3, ..., k_{max}. Notice that in a social network, the maximal hop between two members is not very large, e.g., k_{max}=7 in DBLP dataset. Moreover, for space saving, we only store the neighbors of u_i whose id is greater than i. For example, if u_1 is one 2-hop neighbor of u_2, we do not store u_1 as the 2-hop neighbor of u_2 and we only store u_2 as the 2-hop neighbor of u_1 since they are symmetrical. Before checking the social distance, we only compare which id is greater for two members and choose u_j as the user with the smaller id and u_i as the member with larger id.

Recall the example of checking the social distance of u_3 and u_5 using (c-1)-hop neighbors and reverse c-hop neighbors. We have maintained the top 1-hop neighbors of u_3 (e.g.,its 1-hop neighbors are u_0, u_2, u_4, u_9) and reverse 2-hop neighbors (e.g., its 3-hop neighbor is u_5 and 4-hop neighbor is empty.) where c is 2. We also compare the value of (c-1)=1 with k=3. Because (c-1) $<$ 3, we check whether u_5 is one of 4-hop neighbors of u_3. Since there is no 4-hop neighbors to u_3, we determine that the social distance of u_3 and u_5 is not greater than 3. Compared with using h-hop neighbors in the above example, we can see that checking the social distance using (c-1)-hop neighbors and reverse c-hop neighbors requires less examination than that using h-hop neighbors.

Note that, when storing the hop neighbors, we only store the hop neighbor whose id is greater than the user. For example, we only store 1-hop neighbors of u_3 as $\{u_4, u_9\}$. The 1-hop neighbors of u_0 (which is also one of u_3' 1-hop neighbors) are $\{u_1, u_2, u_3, u_4, u_9, u_{11}\}$, which contains u_3. If we need to check whether u_3 is one of the 1-hop neighbors of u_0 , we first compare the id of u_3 and u_0 and find that u_3 is in the 1-hop neighbors of u_0. This reduces half of space cost.

With regard to the updates for NLRNL, since deleting/inserting one vertex can be divided into edge deletions/insertions, we mainly discuss how to update NLRNL when inserting one new edge or deleting one old edge. For inserting one new edge, it may affect the shortest hops between two vertexes. The main idea is to quickly figure out those affected shortest hops. For each vertex v, we should update its corresponding hop neighbors. Let d be the distance of the new edge to v. For the hop neighbors whose distance to v is less than d+1, their distance to v does not change. That is, m-hop neighbors ($m \leq d+1$) still are m-hop neighbors. For those hop neighbors whose distance to v is greater than d+1, we check whether the vertexes on the corresponding shortest path contain both of the two ending vertexes of the new edge. If they do, we update the shortest hops. Otherwise, the shortest hops do not change. Thus, we should maintain the shortest path for every two nodes. For deleting one edge, we first find those hop neighbors whose shortest path to v goes through this edge. After finding those hop neighbors, we check whether the corresponding shortest path can be updated with this new edge. If so, we update it and insert it into the corresponding hop neighbors.

Space cost of indexes. Let n be the number of vertexes in the social network. For the NL index, we store the 1-hop neighbors, 2-hop neighbors,..., h-hop neighbors for each vertex. Assume that the average number of neighbors for each vertex is m. The number of 2-hop neighbors is m^2, since in the worst case all neighbors of 1-hop neihbors are 2-hop neighbors. Thus, it needs to take O($m + m^2 + ... + m^h$)= O($\frac{m\times(m^h-1)}{m-1}$) space cost for each vertex to store its 1-hop neighbors, 2-hop neighbors, ..., h-hop neighbors. In total, the NL index requires O($\frac{n\times m\times(m^h-1)}{m-1}$)=O($n\times m^h$) space cost. As for the NLRNL index, it stores 1-hop neighbors, 2-hop neighbors, ..., (c-1)-hop neighbors and reverse c-hop neighbors. That is, it does not store the c-hop neighbors, and the size is m^c in the worst case. For each vertex, it takes O($n - m^c - 1$) space cost for those top (c-1)-hop neighbors and reverse c-hop neighbors. Note that, when storing the hop neighbors, we only store the hop neighbor whose id is greater than the user. Therefore, it only takes half of the space cost, i.e., the NLRNL index takes O($\frac{n\times(n-m^c-1)}{2}$)=O($n\times$min$\{m^c, n-m^c\}$).

VI. The Algorithm for Finding Diverse Groups

As shown in the experimental results, the groups returned by KTG-VKC or KTG-VKC-DEG are not diverse. For example, the returned groups of size 3 are sometimes the style like "$u_1u_2u_3$", "$u_1u_2u_4$", and "$u_1u_2u_5$", where the groups are heavily overlapped. If u_1 or u_2 is not available as a member, those groups are all not available. Thus, in this section we extend our algorithm to find the diverse groups.

A. Definition of diversified keyword-based tenuous groups

To address the diversity issue, we use Jaccard distance on the set of the result groups, by considering their common members (vertices), following the ideas of diversified groups in work PerK [35] and DivQ [27]. The diversity function is defined as below.

Definition 9: **Diversity function.** Given two groups g_1 and g_2, we define the diversity score of these two groups, denoted as dL, as follows.

$$dL(g_1, g_2) = \frac{|\ g_1 \cup g_2| - |g_1 \cap g_2|}{|g_1 \cup g_2|}. \tag{2}$$

The diversity score $dL(RG)$ for the query result set RG of size N is the average diversity score for all group pairs, where $\frac{N\times(N-1)}{2}$ is the total number of group pairs in RG, as defined below.

$$dL(RG) = \frac{2 \times \sum_{g_i \neq g_j, \in RG} dL(g_i, g_j)}{N \times (N-1)}. \tag{3}$$

Accordingly, the diversity problem is to find an RG that maximizes $dL(RG)$. Combining with the goal with high query keyword coverage for each group and high diversity between the result groups, we define the totoal score as follows.

$$score(RG) = \gamma \times \min_{g \in RG} QKC(g) + (1-\gamma) \times dL(RG) \tag{4}$$

, where γ controls the contribution of the two components, diversity and keyword coverage.

Based on Equation (5), we define the diversified keyword-based tenuous groups retrieval problem as follows.

Definition 10: **Diversified KTG (DKTG) problem.** Given a query Q with keyword set W_Q, a tenuity constraint parameter k, an integer N, and a tenuous group size p, the DKTG query is to find top N groups set RG that have the highest score(RG).

B. The DKTG-Greedy Algorithm

We develop a new algorithm, called *DKTG-Greedy*, by exploiting a greedy heuristic on top of KTG-VKC-DEG to address the DKTG problem. Initially, DKTG-Greedy greedily finds the result group with the maximal keyword coverage. As such, the current highest keyword coverage C_{max} is initialized with the maximal keyword coverage. The heuristic iteratively finds a feasible group whose keyword coverage is not less than C_{max} in the remaining qualified users set S_R via revising KTG-VKC-DEG to return only one group and removes the result users in existing result groups from S_R until we obtain top N groups. Notice that a feasible group with the highest keyword coverage can make the largest contribution for the keyword coverage part of the total score. Moreover, removing the result users in existing result groups from S_R can make the largest contribution to the diversity part of $score(RG)$.

After obtaining a group with the highest keyword coverage with coverage C_{max}, we first remove those users who are already in RG from the remaining users set S_R. Then we invoke KTG-VKC-DEG to find a new result group whose keyword coverage is not less than C_{max}. If such a new result group is not found, we have two strategies to deal with it. (1) We add one user in the result group one by one into the remaining users set and combine this user with the users in S_R until we find a new result group whose keyword coverage is not less than C_{max}. However, we may not find such a new result group even if all the result users are added into S_R. (2) We keep the new group with keyword coverage C'_{max} (whose keyword coverage is less than C_{max}) as a result group and update it as C_{max}. In DKTG-Greedy, we adopt this strategy.

Example 3: Recall the example in Figure 1 for illustration of the DKTG-Greedy algorithm. Given a DKTG $\langle W_Q = \{SN, QP, DQ, GQ, GD\}$, p=3, k=1, N=2$\rangle$ query, DKTG-Greedy utilizes the valid keyword sorting and degree for sorting. Similar with KTG-VKC-DEG, S_R becomes $\{u_{10}$, u_0, u_{11}, u_1, u_2, u_3, u_7, u_5, u_4 $\}$. Then we invoke the DKTG-Greedy to explore the candidate groups. By selecting a vertex from S_R to S_I iteratively, we first obtain the top-1 result group $\{u_{10}, u_5, u_1\}$ with the query keyword coverage 0.8. Notice that if we take group $\{u_{10}, u_5, u_2\}$ as the 2-nd result group, the diversity score of this results is $dL(RG)$=(4-2)/4=0.5. As mentioned, there is only one reviewer who is different between these two groups, and the diversity is not good. Accordingly, we continue to move forward on the branch S_I= $\{u_0\}$, and S_R becomes $\{u_7\}$. Since there is not enough reviewers with $\{u_0\}$ to be a feasible group, we terminate the branch $\{u_0\}$. In the sequel, branch S_I= $\{u_{11}\}$ is explored, and S_R becomes $\{$ u_7, $u_2, u_3, u_4\}$ since we remove the reviewers inside $\{u_{10}, u_5, u_1\}$ from S_R . By selecting u_7 from S_R according to the order in S_R and S_I becomes $\{u_{11}, u_7\}$. At this time, S_R becomes $\{$ u_2, u_3, $u_4\}$. Similar to the above steps, DKTG-Greedy returns $\{u_{11}$, u_7, $u_2\}$ the 2-nd result group with the query keyword coverage 0.8. The diversity score of the two result groups $\{u_{10}$, u_5, $u_1\}$ and $\{u_{11}$, u_7, $u_2\}$ is $dL(RG)$=(6-0)/6=1. The query processing ends since the diversity score is the maximal score without any repeated reviewers.

C. Theoretical Analysis

To analyze the approximation ratio of our DKTG-Greedy algorithm, we first define the Δ-approximate DKTG problem as follows.

Definition 11: (Δ*-approximate DKTG problem*). Given an DKTG query Q $\langle W_Q$, p, k, $N\rangle$ and a scalar Δ where $\Delta <$ 1, suppose the total score of the optimal result groups for Q is S_{opt}. The Δ*-approximate DKTG problem* is to find an approximate solution with total score S_{appr} such that $\Delta \geq \frac{S_{appr}}{S_{opt}}$.

For the optimal result groups, denoted as R_{opt}, $\min_{g \in R_{opt}} QKC(g)$ is 1, and $dL(R_{opt})$ is also 1. Thus, the corresponding score is $score(R_{opt}) = \gamma \times 1 + (1-\gamma) \times 1$=1. While for our DKTG-Greedy algorithm, assume that the returned result groups, denoted as R_{greedy}, DKTG-Greedy returns the result group which covers at least query one keyword, therefore $\min_{g \in R_{greedy}} QKC(g)$ is $\frac{1}{|W_Q|}$, where $|W_Q|$ is the size of query keywords. Since DKTG-Greedy removes the result users from the remaining users set, thus this makes the largest diversity contribution and $dL(R_{greedy})$ is 1. Accordingly, $score(R_{greedy}) = \gamma \times \frac{1}{|W_Q|} + (1-\gamma) \times 1$= 1+$\gamma$ $\times$ ($\frac{1}{|W_Q|}$ $-$ 1)=1+$\frac{\gamma \times (1-|W_Q|)}{|W_Q|}$. Since $|W_Q|$ is not less than 1, $\frac{\gamma \times (1-|W_Q|)}{|W_Q|}$ is less than 0. Let α denote $\frac{\gamma \times (|W_Q|-1)}{|W_Q|}$. Therefore, we have $score(R_{greedy}) = 1 - \alpha$ and the approximation ratio of DKTG-Greedy is $\frac{score(R_{greedy})}{score(R_{opt})} = 1 - \alpha$.

VII. EXPERIMENTS

In this section, we evaluate the performance of the proposed algorithms for the KTG and DKTG queries. All the algorithms are implemented in C++, while the experiments are conducted on an Intel Core I7 2.3 GHz PC with 120GB RAM.

We conduct experiments on four real datasets.

- ***DBLP***. It consists of 200000 nodes and 1228923 edges.[1]
- ***Gowalla***. It consists of 67320 nodes and 559200 edges.[2]
- ***Brightkite***. It contains 58288 users and 214038 edges.[2]
- ***Flickr***. It consists of 157681 nodes and 1344397 edges.[2]

Table I
PARAMETER RANGES AND DEFAULTS VALUES

Parameters	**Range**
$\sharp$ of group size (p)	3, **4**, 5, 6, 7
$\sharp$ of social constraint (k)	1, **2**, 3, 4
Query keyword size ($\lvert W_Q \rvert$)	4, **5**, 6, 7, 8
N value	3, **5**, 7, 9, 11

We conduct a performance evaluation on the efficiency of the KTG and DKTG algorithms in terms of the latency of the proposed KTG and DKTG algorithms under various parameter settings (summarized in Table I, numbers in bold are the default settings). We measure the average latency as the performance metric corresponding to five different parameters: (a) group size p; (b) query keyword size $|W_Q|$; (c) social constraint k; and (d) the N value for returning top N groups. We randomly generate four groups of queries corresponding to each dataset where each group consists of 100 queries. In each experiment, we test one parameter at a time (by fixing other parameters at their default values). The reported experimental results are obtained by averaging the processing time of queries.

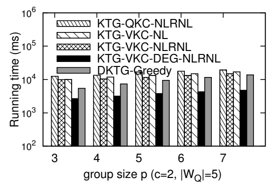
A. Efficiency of the KTG and DKTG Algorithms

In this section, we compare the efficiency of the proposed KTG and DKTG algorithms, including *KTG-QKC-NLRNL* (variant of Algorithm 1 using query keyword coverage as the sorting), *KTG-VKC-NL* (Algorithm 1), *KTG-VKC-NLRNL*, *KTG-VKC-DEG-NLRNL* and *DKTG-Greedy*. QKC denotes the algorihtm uses the query keyword coverage as the sorting, VKC means using the valid keyword coverage as the sorting, while VKC-DEG means the algorithm utilizes both the valid keyword coverage and degree as the sorting. For the data structures to check the social distance, NL means that the algorithm utilizes the h-hop neighbor index to check the social distance and NLRNL means that the algorithm utlizes the (c-1)-hop neighbor and reverse c-hop neighbor index to check the social distance. In DKTG-Greedy, we adopt *KTG-VKC-DEG-NLRNL* in getOneGroup function. Accordingly, we test the four KTG and one DKTG algorithms using different datasets as described above.

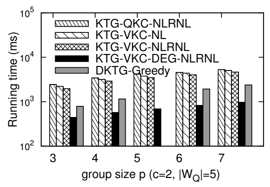
Effect of p value. We first compare the average running time of the four KTG algorithms for processing KTG queries and DKTG-Greedy for processing DKTG queries by increasing the group size p. Figure 3(a) shows the result on the Gowlla dataset. As shown, KTG-VKC-DEG-NLRNL performs much better than KTG-QKC-NLRNL, KTG-VKC-NL and KTG-VKC-NLRNL, while KTG-VKC-NLRNL outperforms KTG-VKC-NL and KTG-QKC-NLRNL. This shows that the KTG-VKC-DEG-NLRNL algorithm giving priority to the combination of valid keyword coverage and degree, is more efficient than that giving priority to valid keyword coverage because the degree sorting allows the algorithm to form a feasible solution early and the pruning improves efficiency. In addition, the algorithm utilizing the NLRNL index is faster than that using the NL index. Since the algorithm using the valid keyword coverage for sorting is more efficient than that using query keyword coverage, we do not show the time of KTG-QKC-NLRNL in the remaining query results. While for the DKTG algorithm, DKTG-Greedy outperforms KTG-VKC-NL and KTG-VKC-NLRNL and is comparable with KTG-VKC-DEG-NLRNL to support the diversity. When the p value increases, the query time becomes longer, as more users need to be examined and the number of combinations becomes larger. The results on other three datasets are similar to that on the Gowalla dataset.

Effect of social constraint k. Figure 4 compares the performance of the three KTG algorithms and DKTG-Greedy by varying k. For the Gowalla dataset, KTG-VKC-DEG-NLRNL outperforms KTG-VKC-NL and KTG-VKC-NLRNL significantly under various k values because KTG-VKC-DEG-NLRNL gives priority to the combination of valid keyword coverage and degree which avoids some invalid group combinations to form a good and feasible group early compared to KTG-VKC-NL and KTG-VKC-NLRNL. In addition, the results shows the superiority of KTG-VKC-NLRNL over KTG-VKC-NL because checking the social distance using NLRNL is more efficient than that using NL. DKTG-Greedy is faster than KTG-VKC-NL and KTG-VKC-NLRNL, and is slower than KTG-VKC-DEG-NLRNL. Moreover, the processing time of all four algorithms increases as the k value increases on Gowalla dataset. This is because there are less valid users whose social distance is greater than k when k becomes larger.

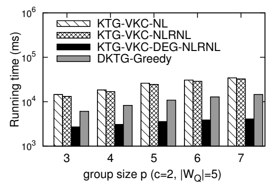
[1]https://github.com/samjjx/pp-data
[2]https://snap.stanford.edu/data/index.html.



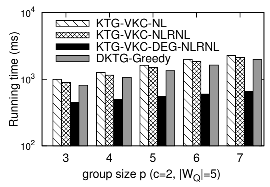
(a) Gowalla dataset



(b) Brightkite dataset

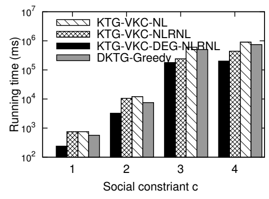


(c) Flickr dataset

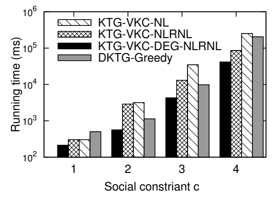


(d) DBLP dataset

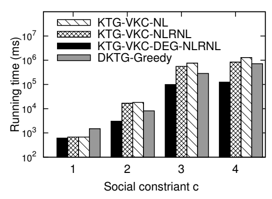
Figure 3. Performance vs. query group size p



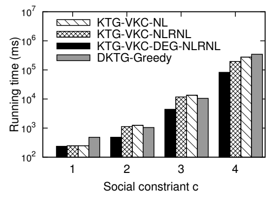
(a) Gowalla dataset



(b) Brightkite dataset

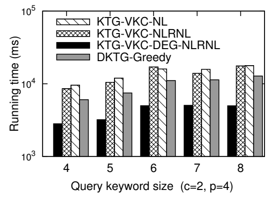


(c) Flickr dataset

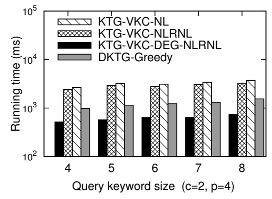


(d) DBLP dataset

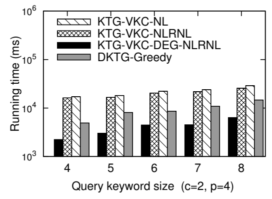
Figure 4. Performance vs. social constraint k



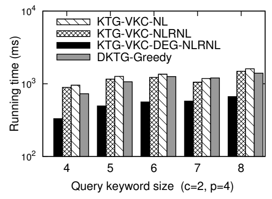
(a) Gowalla dataset



(b) Brightkite dataset

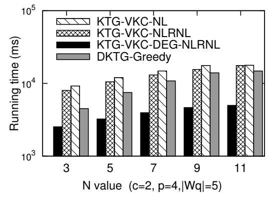


(c) Flickr dataset

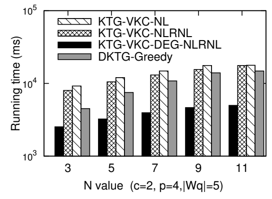


(d) DBLP dataset

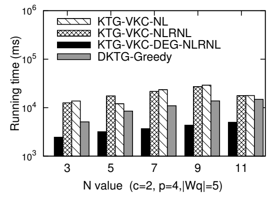
Figure 5. Performance vs. query keyword size



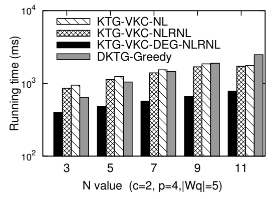
(a) Gowalla dataset



(b) Brightkite dataset



(c) Flickr dataset



(d) DBLP dataset

Figure 6. Performance vs. different N values

The results on the Flickr dataset in Figure 4(c) and DBLP dataset in Figure 4(d) show the similar trend with the result on the Gowalla dataset.

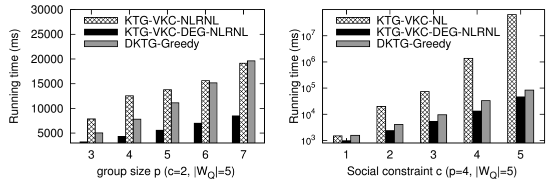
Effect of query keyword size. In addition, in Figure 5, we compare the three KTG algorithms and DKTG-Greedy by varying the query keyword size. If the query keyword size is larger, more users cover at least one of the query keywords. For the *Gowalla* dataset, the results in Figure 5(a) shows the superiority of KTG-VKC-DEG-NLRNL over the KTG-VKC-NL and KTG-VKC-NLRNL algorithms. The processing time of KTG-VKC-DEG-NLRNL takes less than 5000 ms, but KTG-VKC-NL and KTG-VKC-NLRNL take more than 10000 ms, because KTG-VKC-DEG-NLRNL prunes the unqualified groups effectively via the degree sorting to accelerate the query processing. All the algorithms are very stable when the query keyword size becomes larger because all the algorithms have enough qualified users covering the query keywords to form top N groups, i.e., the groups jointly covering all the query keywords. Similar results on Brightkite, Flickr and DBLP dataset are observed in Figures 5(b), 5(c) and 5(d).

Impact on denser graph and large graph. At last, we test our algorithms by varying the group size on one denser graph, i.e., the Twitter dataset[2] where the number of nodes is 81,306 and the number of edges is 1,768,149, and one large DBLP dataset with one million nodes. As shown in Figure 7(a) of Twitter dataset, our KTG-VKD-DEG algorithm outperforms KTG-VKD significantly. From Figure 7(b) by varying the social constraint, KTG-VKC-DEG-NLRNL shows good scalability on the large graph, while KTG-VKC-NL is very slow (up to 16 hours) with a large social constraint c (e.g., 4 and 5).

B. *Effectiveness of the Algorithms*

To demonstrate the effectiveness of our proposed algorithms, we conduct a case study on DBLP dataset. In contrast to the model in TAGQ [18], our models can ensure the tenuity and keyword coverage over the reviewers to facilitate a comprehensive review.

Figure 8 shows a case study on DBLP dataset. The top part shows the detail of the query, the bottom part shows the keywords of result users returned by the three algorithms and



(a) Twitter dataset (b) DBLP dataset with 1M vertexes

Figure 7. Denser graph and Large dataset

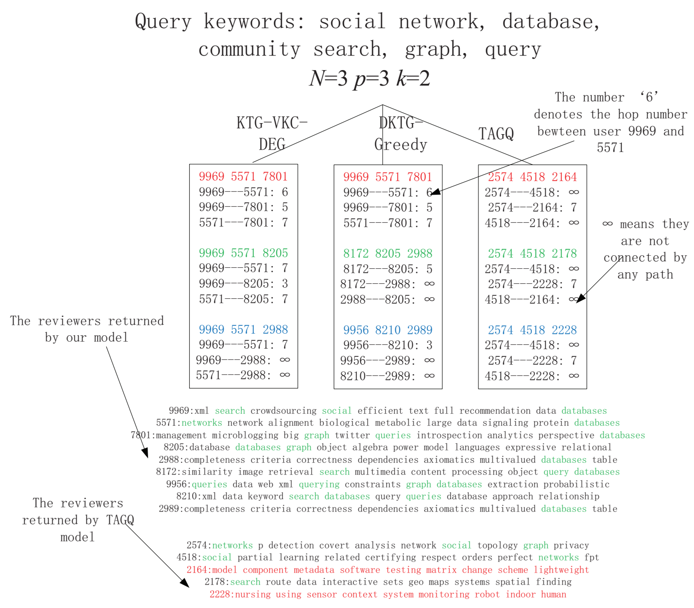

Query keywords: social network, database, community search, graph, query
N=3 p=3 k=2
KTG-VKC-DEG
DKTG-Greedy
TAGQ
The number ‘6’ denotes the hop number bewteen user 9969 and 5571
∞ means they are not connected by any path
The reviewers returned by our model
The reviewers returned by TAGQ model
9969 5571 7801
9969---5571: 6
9969---7801: 5
5571---7801: 7
9969 5571 8205
9969---5571: 7
9969---8205: 3
5571---8205: 7
9969 5571 2988
9969---5571: 7
9969---2988: ∞
5571---2988: ∞
9969 5571 7801
9969---5571: 6
9969---7801: 5
5571---7801: 7
8172 8205 2988
8172---8205: 5
8172---2988: ∞
2988---8205: ∞
9956 8210 2989
9956---8210: 3
9956---2989: ∞
8210---2989: ∞
2574 4518 2164
2574---4518: ∞
2574---2164: 7
4518---2164: ∞
2574 4518 2178
2574---4518: ∞
2574---2228: 7
4518---2164: ∞
2574 4518 2228
2574---4518: ∞
2574---2228: 7
4518---2228: ∞
9969:xml search crowdsourcing social efficient text full recommendation data databases
5571:networks network alignment biological metabolic large data signaling protein databases
7801:management microblogging big graph twitter queries introspection analytics perspective databases
8205:database databases graph object algebra power model languages expressive relational
2988:completeness criteria correctness dependencies axiomatics multivalued databases table
8172:similarity image retrieval search multimedia content processing object query databases
9956:queries data web xml querying constraints graph databases extraction probabilistic
8210:xml data keyword search databases query queries database approach relationship
2989:completeness criteria correctness dependencies axiomatics multivalued databases table
2574:networks p detection covert analysis network social topology graph privacy
4518:social partial learning related certifying respect orders perfect networks fpt
2164:model component metadata software testing matrix change scheme lightweight
2178:search route data interactive sets geo maps systems spatial finding
2228:nursing using sensor context system monitoring robot indoor human

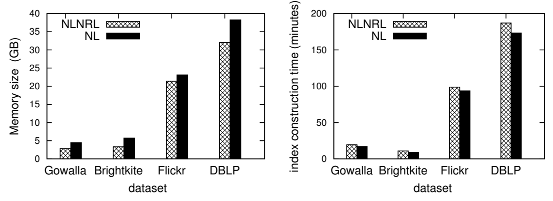
Figure 8. Case Study

the middle part shows the top 3 groups with the hop number for every two reviewers in a group returned by each algorithm. We compare our two algorithms KTG-VKD-DEG and DKTG-Greedy with TAGQ by setting γ=0.5 for equally controlling the contribution of the two components. We can see that the reviewers (e.g., 2,164 and 2,228, as denoted by the red lines) returned by TAGQ do not have any keywords covered by the query keywords. That is because, the model used in TAGQ aims to maximize the average coverage of query keywords for the result group. However, this may lead some users do not have any query keywords coverage. All the users in a result group satisfy the social constraint. In addition, similar to KTG-VKD-DEG, TAGQ also does not support the diversity of the users in the result groups.

C. Space and Construction Overhead of Index

Finally, we compare the space overhead of using NL and NLRNL indexes on all four datasets. Note that for NL index, it stores 1-hop neighbors, 2-hop neighbors, ..., c-hop neighbors for each node[3], while for NLRNL index, it stores not only 1-hop neighbors, 2-hop neighbors, ..., (c-1)-hop neighbors but also (c+1)-hop neighbors, (c+2)-hop neighbors,...h_{max}-hop neighbors (h_{max} is the maximal hop neighbors of this vertex). The results on Figure 9(a) show that the space cost of the NLRNL index is less than that using the NL index. This is because the number of c-hop neighbors is very large and the NL index needs to store two times for the relationship of every two vertexes u_i and u_j, i.e., u_i is one of u_j' x-hop neighbors and u_j is also one of u_i' x-hop neighbors, to expand c-hop neighbors to (c+1)-hop neighbors. Morever, we also test the index construction overhead, as shown in Figure 9(b). The cost of constructing NLRNL takes more time than constructing NL because NLRNL also requires to maintain the reverse c-hop neighbors besides maintaining the (c-1)-hop neihbors for each vertex.

[3] c is the value with the most corresponding hop neighbors.



(a) Space Overhead (b) Index construction Overhead

Figure 9. Space and Construction Overhead

VIII. CONCLUSION

In this paper, we formulate the problem of *keywords based tenous-socially groups* (KTG) query for finding top N tenuous groups which jointly cover the most query keywords. We carry out a systematic study on the KTG query. First, we propose two exact algorithms, namely *KTG-VKC* and *KTG-VKC-DEG*, which give priority to the valid keyword coverage and the combination of valid keyword coverage and degree respectively, to select one user to form a feasible group by adopting a branch and bound (BB) strategy, also using *keyword pruning* and *k-line filtering*. To support the diversity of KTG, we also formalize the problem of *diversified KTG(DKTG)* problem. To address the DKTG problem, we propose the *DKTG-Greedy* algorithm to exploit a greedy heuristic in a combination with KTG-VKC-DEG. Moreover,we design two alternative indexes, namely *NL* and *NLRNL* indexes, to check whether the social distance of any two users is greater than social constraint k in the above algorithms. At last, a comprehensive performance evaluation is conducted to validate the proposed ideas and demonstrate the efficiency and effectiveness of the proposed indexes and algorithms.

Acknowledgements. This work is supported by the National Natural Science Foundation of China (U1911203,U1811264,U1811262,U1811261,U2001211,U22B2060, 61902438, 61902439, 62102463), the Key-Area Research and Development Program of Guangdong Province (2020B0101100001, 2018B01010700), Guangdong Basic and Applied Basic Research Foundation (2019B1515130001, 2019A1515011704), Industry-university-research Innovation Fund for Chinese Universities (No. 2020ITA03009), and Hong Kong RGC Grant C2004- 21GF, 12202221, 12200021, and 12200819. Ningning Cui and Jianliang Xu are the corresponding authors of this work.

References

[1] C.-Y. Shen, L.-H. Huang, D.-N. Yang, H.-H. Shuai, W.-C. Lee, and M.-S. Chen, "On finding socially tenuous groups for online social networks," in *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 2017, pp. 415–424.

[2] W. Li, "Finding tenuous groups in social networks," in *2018 IEEE International Conference on Data Mining Workshops (ICDMW)*, 2018, pp. 284–291.

[3] Y. Li, H. Sun, L. He, J. Huang, J. Chen, H. He, and X. Jia, "Querying tenuous group in attributed networks," *The Computer Journal*, 2020.

[4] C. Y. Shen, H. H. Shuai, D. N. Yang, G. S. Lee, L. H. Huang, W.-C. Lee, and M. S. Chen, "On extracting socially tenuous groups for online social networks with k-triangles," *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–1, 2020.

[5] S. B. Seidman, "Network structure and minimum degree," *Social networks*, vol. 5, no. 3, pp. 269–287, 1983.

[6] 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)*, 2017, pp. 156–163.

[7] M. Sozio and A. Gionis, "The community-search problem and how to plan a successful cocktail party," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2010, pp. 939–948.

[8] 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.

[9] W. Cui, Y. Xiao, H. Wang, and W. Wang, "Local search of communities in large graphs," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, 2014, pp. 991–1002.

[10] S. Ebadian and X. Huang, "Fast algorithm for k-truss discovery on public-private graphs," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, 2019, pp. 2258–2264.

[11] X. Huang, L. V. S. Lakshmanan, J. X. Yu, and H. Cheng, "Approximate closest community search in networks," *Proc. VLDB Endow.*, vol. 9, no. 4, p. 276287, 2015.

[12] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu, "Querying k-truss community in large and dynamic graphs," in *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, 2014, pp. 1311–1322.

[13] J. Hu, R. Cheng, K. C.-C. Chang, A. Sankar, Y. Fang, and B. Y. Lam, "Discovering maximal motif cliques in large heterogeneous information networks," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, 2019, pp. 746–757.

[14] C. Ma, R. Cheng, L. V. Lakshmanan, T. Grubenmann, Y. Fang, and X. Li, "Linc: a motif counting algorithm for uncertain graphs," *Proceedings of the VLDB Endowment*, vol. 13, no. 2, pp. 155–168, 2019.

[15] M. Bougeret, N. Bousquet, R. Giroudeau, and R. Watrigant, "Parameterized complexity of the sparsest k-subgraph problem in chordal graphs," in *International Conference on Current Trends in Theory and Practice of Informatics*. Springer, 2014, pp. 150–161.

[16] C.-Y. Shen, H.-H. Shuai, D.-N. Yang, Y.-F. Lan, W.-C. Lee, P. S. Yu, and M.-S. Chen, "Forming online support groups for internet and behavior related addictions," in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, 2015, pp. 163–172.

[17] B.-Y. Hsu, Y.-F. Lan, and C.-Y. Shen, "On automatic formation of effective therapy groups in social networks," *IEEE Transactions on Computational Social Systems*, vol. 5, no. 3, pp. 713–726, 2018.

[18] Y. Li, H. Sun, L. He, J. Huang, J. Chen, H. He, and X. Jia, "Querying Tenuous Group in Attributed Networks," *The Computer Journal*, 2020.

[19] S. Cheng, M. Chrobak, and V. Hristidis, "Slowing the firehose: Multidimensional diversity on social post streams," in *Proceedings of the 19th International Conference on Extending Database Technology, EDBT 2016, Bordeaux, France, March 15-16, 2016, Bordeaux, France, March 15-16, 2016*, 2016, pp. 17–28.

[20] G. J. Fakas, Z. Cai, and N. Mamoulis, "Diverse and proportional size-l object summaries for keyword search," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, 2015, pp. 363–375.

[21] M. Hasan, A. Kashyap, V. Hristidis, and V. J. Tsotras, "User effort minimization through adaptive diversification," in *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*. ACM, 2014, pp. 203–212.

[22] M. R. Vieira, H. L. Razente, M. C. N. Barioni, M. Hadjieleftheriou, D. Srivastava, C. T. Jr., and V. J. Tsotras, "On query result diversification," in *Proceedings of the 27th International Conference on Data Engineering, ICDE 2011, April 11-16, 2011, Hannover, Germany*, 2011, pp. 1163–1174.

[23] J. Yang, W. Yao, and W. Zhang, "Keyword search on large graphs: A survey," *Data Sci. Eng.*, vol. 6, no. 2, pp. 142–162, 2021.

[24] Marina, Drosou, Evaggelia, and Pitoura, "Search result diversification," *Sigmod Record Acm Sigmod*, 2010.

[25] L. Lin, P. Yuan, R. Li, and H. Jin, "Mining diversified top-r lasting cohesive subgraphs on temporal networks," *IEEE Transactions on Big Data*, 2021.

[26] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang, "Selecting stars: The k most representative skyline operator," in *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*, 2007, pp. 86–95.

[27] E. Demidova, P. Fankhauser, X. Zhou, and W. Nejdl, "*DivQ*: diversification for keyword search over structured databases," in *Proceeding of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2010, Geneva, Switzerland, July 19-23, 2010*. ACM, 2010, pp. 331–338.

[28] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong, "Diversifying search results," in *Proceedings of the Second International Conference on Web Search and Web Data Mining, WSDM 2009, Barcelona, Spain, February 9-11, 2009*, 2009, pp. 5–14.

[29] A. Angel and N. Koudas, "Efficient diversity-aware search," in *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2011, Athens, Greece, June 12-16, 2011*, 2011, pp. 781–792.

[30] W. Fan, X. Wang, and Y. Wu, "Diversified top-k graph pattern matching," *Proc. VLDB Endow.*, vol. 6, no. 13, pp. 1510–1521, 2013.

[31] L. Yuan, L. Qin, X. Lin, L. Chang, and W. Zhang, "Diversified top-k clique search," in *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015*, 2015, pp. 387–398.

[32] R. Zhu, Z. Zou, and J. Li, "Fast diversified coherent core search on multi-layer graphs," *VLDB J.*, vol. 28, no. 4, pp. 597–622, 2019.

[33] F. Zhang, X. Lin, Y. Zhang, L. Qin, and W. Zhang, "Efficient community discovery with user engagement and similarity," *VLDB J.*, vol. 28, no. 6, pp. 987–1012, 2019.

[34] M. Kargar, A. An, and X. Yu, "Efficient duplication free and minimal keyword search in graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 7, pp. 1657–1669, 2014.

[35] K. Stefanidis, M. Drosou, and E. Pitoura, "Perk: personalized keyword search in relational databases through preferences," in *EDBT 2010, 13th International Conference on Extending Database Technology, Lausanne, Switzerland, March 22-26, 2010, Proceedings*, ser. ACM International Conference Proceeding Series, vol. 426. ACM, 2010, pp. 585–596.

[36] Z. Cai, G. Kalamatianos, G. J. Fakas, N. Mamoulis, and D. Papadias, "Diversified spatial keyword search on RDF data," *VLDB J.*, vol. 29, no. 5, pp. 1171–1189, 2020.

[37] M. Zhang, L. Li, W. Hua, and X. Zhou, "Efficient 2-hop labeling maintenance in dynamic small-world networks," in *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April 19-22, 2021*, 2021, pp. 133–144.