Multi-attributed Community Search in Road-social Networks

Fangda Guo[†], Ye Yuan[§], Guoren Wang[§], Xiangguo Zhao[†], Hao Sun[†]
[†]*School of Computer Science and Engineering, Northeastern University, Shenyang, China*
[§]*School of Computer Science and Technology, Beijing Institute of Technology, Beijing, China*
{[†]fangda@stu, [§]yuanye@, [†]zhaoxiangguo@, [†]andynosh@stu}mail.neu.edu.cn, [§]wanggr@bit.edu.cn

***Abstract*—Given a location-based social network, how to find the communities that are highly relevant to query users and have top overall scores in multiple attributes according to user preferences? Typically, in the face of such a problem setting, we can model the network as a multi-attributed road-social network, in which each user is linked with location information and d (≥ 1) numerical attributes. In practice, user preferences (i.e., weights) are usually inherently uncertain and can only be estimated with bounded accuracy, because a human user is not able to designate exact values with absolute precision. Inspired by this, we introduce a normative community model suitable for multi-criteria decision making, called multi-attributed community (MAC), based on the concepts of k-core and a novel dominance relationship specific to preferences. Given uncertain user preferences, namely, an approximate representation of weights, the MAC search reports the exact communities for each of the possible weight settings. We devise an elegant index structure to maintain the dominance relationships, based on which two algorithms are developed to efficiently compute the top-j MACs. The efficiency and scalability of our algorithms and the effectiveness of MAC model are demonstrated by extensive experiments on both real-world and synthetic road-social networks.**

I. Introduction

Numerous real-world networks (e.g., social networks) are made up of community structures, where discovering them is an essential problem in network analysis. Recently, community search, which is a kind of query-dependent community discovery problem and is designed to find densely connected subgraphs containing query vertices, has drawn much attention among database professionals due to an ever-growing number of applications [1]–[8].

At the same time, with the prevalence of GPS-enabled mobile devices, location-based social networks (LBSN) are becoming more diverse and complex in recent years (e.g., Facebook Places, Foursquare). Since not only users and friendships are included, but each user is often associated with various properties, such as location information and numerical attributes. The location information is able to bridge the gap between virtual and physical worlds, while the numerical attributes obtained from user profiles or statistical information derived by various network analytics (e.g., influence, similarity, etc.) can characterize the user. For instance, in a scientific collaboration network such as Aminer, every author may have own (spatial) position/address and several numerical attributes (e.g., h-index, #publications, activeness, diverseness, etc.). Typically, we can model such network data as a multi-attributed road-social network, in which each vertex is linked with location and d (≥ 1) numerical attributes.

Given a multi-attributed road-social network, how to identify the query-user-involved communities that are not *dominated* by the other communities according to user's preferences for d numerical attributes? For instance, considering h-index and activeness in the Aminer network, how to find a group of collaborators who are related to and close to the query users and take the activeness in their research fields in recent years as the main criterion? Similarly, considering #publications and diverseness, how to find a community of query users in the Aminer network such that the members provide a good tradeoff in terms of the number of publications and diverse research interests? It is noting that both the social and spatial cohesiveness of groups are incorporated.

In this regard, we want to develop a normative community model suitable for multi-criteria decision making. Considering that the traditional top-j query receives a dataset of records with d-dimensional attributes and a weight vector w of d values assigning the relative importance of each dimension to the user as input, where the weighted sum of attribute values is used as the score of a record, we evolve such scoring method into a multi-attributed community model. Similarly, the weight vector w denotes the user preferences and is therefore crucial in generating useful recommendations for the community. Generally, w can be directly input by the user or mined from his/her past behavior or choices [9], [10].

Driven by the fact that weight accuracy plays a vital role in the applicability and practicality of top-j queries, we argue that the assumption that exact values of a weight vector are known is inherently inaccurate and almost unrealistic. To illustrate this, consider the case of manually assigning weights to the above example. By taking activeness as the main criterion, a user may specify weights 0.2 and 0.8 for h-index and activeness, respectively. At this point, leaving aside the participation of query users and spatial cohesiveness, the weighted sum of these two attribute values can be regarded as the influence (or importance) of the vertex in [4]. On the other hand, based on pure intuition, she may also specify the same weight per dimension (e.g., 0.5 each). The results may be similar to the skyline community [8], since weights are not biased towards any dimension (as verified in our experiments, the skyline community [8] is usually contained in our results). However, it is impractical and unfair for the user to require

weights with absolute precision even a slight variation in the weight (e.g., from 0.2 to 0.19; see Example 3) may remarkably change the results. On the contrary, it would be preferable to take user inputs as generic instructions and leave room for inaccuracies in weight setting. Similarly, for the weight vector computed by preference learning techniques, it should serve only as a rough guide instead of an accurate expression of user preferences. Naturally, this issue can be dealt with by expanding the weight vector to a region[1] and returning all promising results to the user, thus providing a practical and more user-centric design.

Prior work on community search problem has never incorporated the uncertainty of weight vector, thus all existing community search algorithms are unable to answer the above questions. To adequately characterize such interesting communities w.r.t. user preferences, we propose a novel community model called multi-attributed community (MAC) based on the concepts of k-core [12] and r-dominance (variant of traditional sense) [13]. An MAC is a maximal connected k-core with query vertices contained and spatial cohesiveness satisfied, that is not *r-dominated* by other connected k-cores in terms of d-dimensional attributes w.r.t a region of interest R (see Section II for detailed definition). Importantly, since the scores of communities all depend on w, they are necessarily correlated and vary together as w freely lies inside region R. As a result, a partitioning of R forms the output, in which each partition is associated with the MACs when w falls anywhere in that partition. The MAC model is also applicable to many interesting applications, some of which are introduced below.

Personalized optimum community search. In daily life, people always want to discover the optimum community based on different needs. For example, a coach hopes to reorganize the school basketball team around certain players (as query users) to improve offense. In this application, we can limit the query scope to the school and extract three numerical attributes for each player: points, rebounds and assists. By setting the region of user preferences, we can obtain corresponding communities that are not r-dominated by the others. Similar query may also help organizations to analyze customer orientation or perform marketing/promotion activities.

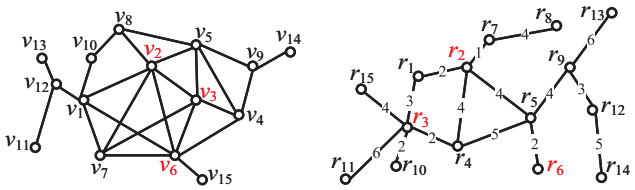
Cohesive groups discovery in LBSNs. In some cases, one may wish to circle the target range by finding cohesive groups to achieve identification, such as COVID-19 precaution and suspect investigation. Given several confirmed cases, possible cases are likely to be within a certain range of them (providing possibility of close contact), and the Jaccard similarity (e.g., hobbies, interests) and influence (e.g., #neighbors) for each user can be extracted as numerical attributes; for the preliminary investigation of a case, given the victim and escape range, motive and #criminal records of the same or different types can be used as numerical attributes. By mining the MACs, we can get the desired cohesive groups related to query users.

Contributions. Efficient solutions are formulated and provided to find multi-attributed communities in road-social networks. Below, we summarize the contributions of this paper.

Novel community model. The MAC model is proposed in road-social networks, which can be used for finding communities not r-dominated by others. To the best of our knowledge, our model is the first to incorporate uncertainty of weight vectors, and our work is also the first to introduce the dominance relationship specific to preferences for community modeling.

New algorithms. An efficient global search, DFS-based algorithm, is first developed to find the top-j MACs for each of the possible weight settings in user preferences. The time and space complexity of DFS-based algorithm are bounded by $O(n'^{2d})$ and $O(m'+n'+n'^{2}\cdot d)$ respectively, where n' and m' denote the number of vertices and edges in the maximal (k,t)-core of a road-social network. To further accelerate the search, we propose a more efficient local search framework, of which two striking features are that (1) its time complexity is much lower than that of global search (at least one order of magnitude faster in practice), and it can find all non-contained MACs in most cases; and (2) it enables the MACs to be output progressively during execution, which is very beneficial to applications expecting only part of the MACs.

Extensive experiments. To demonstrate the high efficiency and scalability of our proposed algorithms and to evaluate the effectiveness of the MAC model, we conduct extensive experiments and a comprehensive case study on both real-world and synthetic road-social networks, in which many significant and interesting communities are able to be discovered.



(a) Social network (G_s) (b) Road network (G_r)

Fig. 1. Example of road-social network

II. Problem Statement

In this section, we formally introduce the multi-attributed community in road-social networks and its search problems. Table I summarizes the notations used throughout this paper.

A. Preliminaries

Road network. We model a road network as an undirected weighted graph $G_r = (V_r, E_r)$, where V_r (resp. E_r) is the sets of vertices (resp. edges). A vertex $u \in V_r$ represents a road intersection/end. An edge $(u,v) \in E_r$ represents a road segment allowed to travel between vertices u and v, and is associated with a non-negative weight $\omega(u,v)$ that represents its cost (e.g., distance/travel time). Let p be a spatial point lying on edge (u,v), and $\omega(u,p)$ be proportional to the length from u to p. By $dist(p,p')$ we refer to the network distance between points p and p' in G_r, which is the sum of edge weights along the least costly (i.e., shortest) path from p to p'.

Social network. We model a social network with d numerical attributes as an undirected graph $G_s = (V_s, E_s, L, X)$, where

[1]There are already preference learning techniques (e.g., [11]) to generate such a region instead of a specific weight vector.

TABLE I
SUMMARY OF NOTATIONS

Notation	Description
Q, j	query vertices and number of MACs to select among
k, t	coreness threshold and query distance threshold
d, R	dimensionality of attributes and preference domain
$dg_H(v), \delta(H)$	degree of vertex v in H and minimum degree of H
$N(v, H)$	neighbor set of vertex v in subgraph H
$L(v)$	mapping of user v's location (i.e., spatial point) in G_r
$X(v)$	d-dimensional vector with real values of user v
$D_Q(H), S(H)$	query distance and score of subgraph H resp.
H_k^t, G_d	the maximal (k,t)-core and r-dominance graph
G_e, G_c	subgraphs of G_d induced by V_H and $V_{G_d} \backslash V_H$ resp.
$l_b(G_e), l_t(G_c)$	vertices in the bottom layer of G_e and top layer of G_c

$V_s(|V_s| = n)$ and $E_s(|E_s| = m)$ denote the sets of vertices (i.e., users) and edges (i.e., social relations) respectively, L and $X(|X| = n)$ are the sets of mappings and d-dimensional vectors defined on V_s respectively. Specifically, for each vertex $v \in V_s$, $L(v)$ provides a mapping of each user's location (i.e., spatial point) in the road network; and v is also associated with a d-dimensional vector with real values, denoted by $X(v) = (x_1^v, \ldots, x_d^v)$, where $X(v) \in X$ and $x_i^v \in \mathbb{R}$.

Road-social network. A road-social network is a pair of graphs, denoted by (G_r, G_s), where G_r is a road network and G_s is a social network. Each user $u_s \in G_s$ is associated with a spatial point p in G_r, i.e., $L(u_s) = p$, indicating the current location of u_s. We assume that p can either be on a vertex or edge of G_r.

For example, Fig. 1 displays a road-social network, where vertices represent users and road junctions, and edges represent social relations and road segments, respectively. For simplicity, the location of user v_i in Fig. 1(a) is on the vertex r_i of the road network in Fig. 1(b). Fig. 2(a) shows the values of part of vertices in three different dimensions.

B. (k, t)-Core

We introduce a novel densely connected substructure, called (k,t)-core, by focusing on the following structural cohesiveness and communication cost.

Structural cohesiveness. In order to model the structural cohesiveness, the generally used k-core model is adopted to indicate the communities [1], [2], [4], [12], [14]. In particular, by $dg_{G_s}(v)$ we refer to the degree of vertex v in social network G_s. Let $H = (V_H, E_H, L_H, X_H)$ be an induced subgraph of G_s, where $V_H \subseteq V_s$, $E_H = \{(u,v) | u, v \in V_H, (u,v) \in E_s\}$, $L_H = \{L(v) | v \in V_H\}$ and $X_H = \{X(v) | v \in V_H\}$.

Definition 1: (k-core.) Given a graph G_s and an integer k, H is a k-core of G_s if each vertex $v \in V_H$ has a degree at least k, i.e., $dg_H(v) \geq k$.

The maximal k-core is the one satisfying that no super k-core containing it exists. We refer to the maximal k in all k-cores containing vertex $v \in V_s$ as the core number of v. In order to avoid confusion, we denote a connected k-core as a k-$\widehat{core}$, since the maximal k-core is not necessarily connected.

Remarks. Although we use k-core as the structural cohesiveness metric, our techniques can also be applied to other criteria such as k-clique [15] and k-truss [3].

Communication cost. For two users $u, v \in G_s$, the length of the shortest path between their locations in G_r is denoted by $dist(L(u), L(v))$, which is equal to $+\infty$ if $L(u)$ and $L(v)$ are not connected. To model the communication cost in G_r, we utilize the notion of query distance below.

Definition 2: (Query Distance.) Given a graph H and query vertices $Q \subseteq V_H$, $\forall q \in Q$, the query distance of $v \in V_H$ is the maximum length of the shortest path from $L_H(v)$ to $L_H(q)$ in G_r, denoted by $D_Q(v) = \max_{q \in Q} dist(L(v), L(q))$; the query distance of H is defined as $D_Q(H) = \max_{u \in V_H} D_Q(u) = \max_{u \in V_H, q \in Q} dist(L(u), L(q))$.

By query distance $D_Q(H)$, the communication cost between query vertices Q and the members in H can be measured. In general, a good community is considered to own a low communication cost, i.e., small $D_Q(H)$. Consider the query vertices $Q = \{v_2, v_3, v_6\}$ in Fig. 1. The query distance of v_7 is $D_Q(v_7) = dist(r_7, r_6) = 7$. The query distance of the subgraph induced by $\{v_2, v_3, v_6, v_7\}$ is equal to $dist(r_3, r_6) = 9$. In the following, we propose a new notion of (k,t)-core, by adapting the concepts of k-core and query distance, to capture dense structural cohesiveness and low communication cost.

Definition 3: ((k,t)-core.) Given graphs (G_r, G_s), query vertices Q, and numbers k and t, H is a (k,t)-core iff H is a k-$\widehat{core}$ of G_s containing Q and $D_Q(H) \leq t$ in G_r.

For a (k,t)-core, its structural cohesiveness increases with k, while proximity to query vertices decreases with t. For instance, in Fig. 1, the subgraph induced by $\{v_2, v_3, v_6, v_7\}$ for $Q = \{v_2, v_3, v_6\}$ is a (k,t)-core with $k = 3$ and $t = 9$.

C. Score of Multi-Attributes

Note that generalizing the existing community models, most of which are only for 1-dimensional attribute such as influence [4], Euclidean distance [6] or keyword similarity [5], [7], [16], [17], to a comparable one with multi-dimensional attributes is nontrivial. Unlike the above, comparing two communities becomes quite tough if either can have d ($d > 1$) values due to d different dimensions. On the other hand, existing multi-dimensional model [8] cannot compare the pros and cons of any skyline communities and make trade-offs based on user preferences. To overcome the above issues, we introduce the *r-dominance* relationship between two communities, that will be developed for defining our multi-attributed community model.

As each vertex $v \in V_s$ associated with a vector $X(v)$, the attributes define a d-dimensional *data domain*. We assume that for each attribute a higher value is preferable, and a spatial index (e.g., R-tree [18]) is used to organize the vector set X.

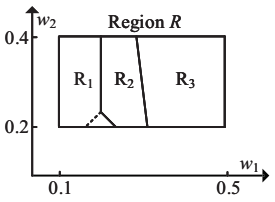
Referring to the traditional top-j queries, the score of a vertex v w.r.t $X(v)$ can also be derived by inputting a weight vector $w = (w_1, w_2, \ldots, w_d)$ as

$$S(v) = \sum_{i=1}^{d} w_i \cdot x_i^v. \quad (1)$$

Thus, the top-j results consist of the j vertices with the highest scores. We assume w.l.o.g. that $w_i \in (0,1)$ for $\forall i \in [1,d]$ and $\sum_{i=1}^{d} w_i = 1$. Such conditions cannot restrain user preferences, because score ranking depends only on the direction of w

User	x_1	x_2	x_3
v_1	8.8	3.6	2.2
v_2	5.9	6.2	6.0
v_3	2.8	5.6	5.1
v_4	9.0	3.3	3.4
v_5	5.0	7.6	3.1
v_6	5.2	8.3	4.3
v_7	2.1	5.0	5.1

(a) 3-dimensional vectors



(b) Pref. domain and MACs

Fig. 2. Numerical attributes and MACs in R

instead of its magnitude [19]; but they allow w to drop one weight (i.e., $w_d = 1 - \sum_{i=1}^{d-1} w_i$), thereby mapping the domain of w to a $(d-1)$-dimensional space, named the *preference domain*. This dimensionality reduction is critical [20], since the dimensionality directly determines the processing time of the costliest operations in our techniques. In the following, w refers to the $(d-1)$-dimensional form of the weight vector. For example, given weights 0.2 and 0.3 for x_1 and x_2 respectively in Fig. 2(a), we have $w_3 = 0.5$, and $S(v_7) = 4.47$.

Community score. Let $H = (V_H, E_H, L_H, X_H)$ be an induced subgraph of G_s. Given a weight vector w, we define score of H as

$$S(H) = \min_{v \in V_H} \{S(v)\}. \quad (2)$$

Here, we have a brief discussion on why the "min" operator in Eq. 2 is used to define $S(H)$. The intention is the same as that in [4]. By using it, all the members in H can be ensured to have a score in terms of d dimensions no less than $S(H)$. In other words, if $S(H)$ is large, vertices in H may not have large values in each dimension but the weighted sum of their attribute values must be large. For instance, consider the previous case of taking the activeness as the main criterion (e.g., weight 0.8) in the collaboration network. Obviously, we can apply the $S(H)$ defined above to quantify the overall quality focusing on the activeness of a group of collaborators.

Definition 4: (r-dominance.) Given a region R in the preference domain, a community H r-dominates another community H' when $S(H) \geq S(H')$ for any weight vector in R, denoted by $H \succ H'$; and $v \succ v'$ denotes that vertex v r-dominates another vertex v'.

Intuitively, in the traditional sense of dominance [21], [22], for any weight vector w the dominator is always superior to the dominee, while r-dominance is specific to weight vectors in R. Although a community may not dominate another based on the skyline community model [8], it might always have a higher score as w is bounded in R. For ease of representation, we assume that R is a hyper-rectangle parallel to axes, but our techniques is directly applicable to general convex polytopes.

For example, Fig. 2(b) illustrates a preference domain (i.e., the domain of weight vector) with $d = 3$, where axis w_1 (resp. w_2) corresponds to the weight for x_1 (resp. x_2) on a scale of 0 to 1, and the region R is a convex polygon (i.e., an axis-parallel rectangle $[0.1, 0.5] \times [0.2, 0.4]$) representing the expanded or approximated user preferences.

D. The MAC Problem

Combining the notion of (k,t)-core and community score, we define the multi-attributed community (MAC) as follows.

Definition 5: (MAC.) Given graphs (G_r, G_s), query vertices $Q \subseteq V_s$, two numbers k and t, and a region of interest R, H is a multi-attributed community in the road-social network, if H satisfies the following conditions:

1. H is a (k,t)-core containing Q.
2. There does not exist an induced subgraph H' ($V_{H'} \supset V_H$) such that H' is a (k,t)-core and $H' \succ H$.

In terms of structural cohesiveness and communication cost, condition (1) requires not only that the community with query vertices Q contained is densely connected, but also that each vertex is spatially close to Q. In terms of maximality and multi-attributed community score, condition (2) ensures that the community is maximal and having the greatest score. The following example illustrates Definition 5.

Example 1: Consider (G_r, G_s), numerical attributes and a region R in Fig. 1 and 2, respectively. Suppose, for instance, that $Q = \{v_2\}$, $k = 2$ and $t = 9$. By Definition 5, the subgraph induced by $\{v_2, v_3, v_5, v_6, v_7\}$ is an MAC with community score equal to $S(v_7)$ if w freely lies anywhere in the upper-left part of R_1 (divided by dotted line), as it meets both conditions. Note that the subgraph induced by $\{v_2, v_3, v_5, v_7\}$ is not an MAC. This is because it is contained in the MAC induced by $\{v_2, v_3, v_5, v_6, v_7\}$ with the same community score, thus fails to satisfy condition (2). □

For most practical applications, we typically tend to focus on the query-user-involved communities, which score higher than (i.e., r-dominate) all other communities in the preference domain R. In this paper, we aim to efficiently discover such communities in road-social networks. Below, two multi-attributed community search problems are formulated.

Problem 1. Given graphs (G_r, G_s), query vertices $Q \subseteq V_s$, two numbers k and t, and a region of interest R, the problem is to find the top-j MACs with the highest community score for each possible weight vector in R. Although the possible weight vectors are infinite in R, a partitioning of R can form the output, in which each partition is associated with the top-j MACs when w falls anywhere in that partition.

For Problem 1, an MAC may be contained in another MAC in the top-j results.

Example 2: Assume that $Q = \{v_2, v_3, v_6\}$, $k = 3$ and $t = 9$ in Fig. 1. The top-2 MACs are subgraphs H_1 and H_2 induced by $\{v_2, v_3, v_6, v_7\}$ and $\{v_2, \ldots, v_7\}$ for any weight vector in R_1, respectively. $S(H_1) = S(v_7)$, and $S(H_2) = S(v_4)$ or $S(v_5)$ on either side of the dotted line. Clearly, H_2 contains H_1. □

To eliminate the containment relations in the top-j results, we study another problem of finding the non-contained MAC.

Definition 6: (non-contained MAC.) An MAC H is a non-contained MAC if there does not exist an induced subgraph H' ($V_{H'} \subset V_H$) such that H' is a (k,t)-core and $H' \succ H$.

The following example illustrates Definition 6.

Example 3: Let us reconsider Example 2. By Definition 6, subgraphs H_1 and H_3 (induced by $\{v_2, \ldots, v_6\}$) are the non-contained MACs for any weight vector in R_1 and in $R_2 \cup R_3$, respectively. For instance, H_3 (resp. H_1) is the top-1 result when $w = (0.2, 0.3)$ (resp. $w = (0.19, 0.3)$). However, H_2 is not a non-contained MAC as it contains H_1 and $H_1 \succ H_2$ for any weight vector in R_1. □

Problem 2. Given graphs (G_r, G_s), query vertices $Q \subseteq V_s$, two numbers k and t, and a region of interest R, the problem is to find the non-contained MAC for every possible weight vector in its corresponding partitioning of R.

Discussions. Another three possible operators "max", "sum" and "avg" are not appropriate for community score. The first two are monotonic w.r.t. the size of community, that is, a community scores higher than its sub-communities. Hence, the answer is always the maximal (k,t)-core, which is independent of numerical attributes X and region R. The last one may cause outliers in the answer, e.g., only a few vertices score very high while the rest score low, resulting in a higher community score. Obviously, this is not an ideal community.

Challenges. Solving the above two problems faces three major challenges. First, the number of k-$\widehat{core}$s containing Q in a multi-attributed network G_s can be exponentially large (even regardless of the query distance in G_r). Thus, enumerating all the k-$\widehat{core}$s to identify the MACs is intractable. Second, unlike traditional top-j queries [23], the score of a community may vary greatly at different parts of R, making it nontrivial to draw conclusions about r-dominance relationships between communities. Third, the MAC model enables a more flexible way to express user preferences in community search problem, which means that inherent inaccuracies in weight specification need to be taken into account. In consequence, without enumerating all the (k,t)-cores, devising an efficient algorithm to detect the MACs is challenging. To overcome these challenges, we will develop efficient algorithms in the following sections.

III. Warming Up for Our Methods

According to Definition 5, regardless of maximality and community score, the multi-attributed communities have to satisfy the constraints of structural cohesiveness and communication cost. Thus, we give two useful lemmas as follows.

Lemma 1: For a number t, the vertices of G_s with query distance greater than t in G_r cannot exist in any MAC.

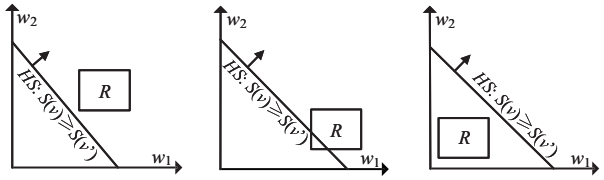
Lemma 2: For an integer k, the MACs must be contained in the maximal k-$\widehat{core}$ containing Q.

The correctness of above lemmas can be verified by Definition 2 and the maximality of k-core, resulting in Lemma 3.

Definition 7: (Maximal (k,t)-core.) For graphs (G_r, G_s) and query vertices Q, the maximal (k,t)-core is a (k,t)-core such that no super (k,t)-core contains it, denoted by H_k^t.

Lemma 3: For two numbers k and t, the MACs must be contained in the maximal (k,t)-core.

Referring to Lemma 3, for a given k and t, we first filter out the vertices of G_s that do not satisfy the query distance threshold t by range query in G_r, which can be accelerated by G-tree [24] or G*-tree [25], to obtain the induced connected subgraph G_s' by the remaining vertices. Next, we do k-core decomposition [14] on the filtered social subgraph G_s' to compute the maximal k-$\widehat{core}$ containing Q. It is noting that we employ the upper bound of coreness [2] before core decomposition. If k is larger than $\lfloor \frac{1+\sqrt{9+8(|E_s'|-|V_s'|)}}{2} \rfloor$, we immediately know there is no k-$\widehat{core}$ w.r.t Q. So far, the maximal (k,t)-core has been found such that the MACs can be obtained through in-depth computation. For example, the maximal $(3,9)$-core, i.e., H_3^9, for $Q = \{v_2, v_3, v_6\}$ is the subgraph induced by $\{v_1, \ldots, v_7\}$, as shown in Fig. 4(a).



(a) v r-dominates (b) r-incomparable (c) v r-dominated

Fig. 3. Cases of r-dominance for vertices v and v'

After abandoning the scheme of enumerating all the (k,t)-cores whose number can be exponentially large even in H_k^t, intuitively, we may think of iteratively deleting the smallest-score vertex w.r.t. d-dimensional attributes until the resulting graph does not have a k-$\widehat{core}$ containing Q. However, at this time we will face another problem, that is, which vertex has the smallest score? To address this issue, we design an effective data structure and construction algorithm to preserve r-dominance relationships between vertices.

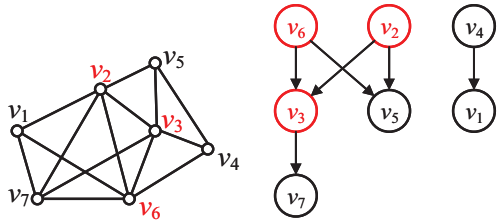
IV. R-Dominance Graph

In this section, we exploit the r-dominance graph to preserve pair-wise r-dominance relationships between vertices in H_k^t, which will be used in our proposed search algorithms.

A. R-Dominance Test

Consider two vertices v and v' where none dominates the other in terms of traditional dominance, that may not draw a reliable conclusion about which vertex ranks higher. Nevertheless, given a preference domain, the inequation $S(v) \geq S(v')$ (resp. equation $S(v) = S(v')$) corresponds to a half-space (resp. hyperplane), of which there are three different cases regarding the positioning against R [13]. Specifically, in Fig. 3(a), v r-dominates v' since half-space $HS : S(v) \geq S(v')$ completely covers R, which means v scores higher for $\forall w \in R$; the case in Fig. 3(c) is symmetric. In Fig. 3(b), v scores higher in one part of R but lower in another, which is called *r-incomparable* as none r-dominates the other.

Clearly, the cases in Fig. 3(a) and 3(c) allow r-dominance conclusions to be safely drawn. In this way, we can determine r-dominance by detecting whether all *polygon vertices* defining R fall into the half-space $HS : S(v) \geq S(v')$. If so (resp. not), v r-dominates (resp. is r-dominated by) v'. Otherwise, v and v' are r-incomparable. The inclusion detecting of each polygon vertex costs $O(d)$, so the r-dominance test requires $O(pd)$ in total, where p is the number of polygon vertices defining R.



(a) H_3^9 for $Q = \{v_2, v_3, v_6\}$ (b) DAG G_d

Fig. 4. The maximal (k, t)-core and r-dominance graph

B. Pair-Wise R-Dominance Relationship

The computation of r-dominance relationships is somewhat similar to that of the k-skyband (i.e., *BBS* [26]), but differs as follows. (1) Rather than traditional dominance, we employ r-dominance and apply its test described in Section IV-A both for vertex-to-vertex and vertex-to-MBB (i.e., minimum bounding box) dominance testing. (2) Due to the fact that w is bounded in R, we adopt a unique sorting key for R-tree nodes (represented by MBB's upper-right corner) and vertices in the heap to accelerate search convergence by leading it to r-dominate as many members as possible first. (3) We preserve the r-dominance relationships between vertices in H_k^t instead of only the top-j layers. It is noting that a max-heap is utilized in our adapted *BBS* and its sorting key is the score of R-tree node/vertex w.r.t. a pivot vector of R, whose value of each dimension is the mean of the polygon vertices of R in that dimension [13]. The correctness of our adaptation is guaranteed as follows. (1) The pivot vector must lie in R due to R's convexity [27]. (2) Vertices popped after v cannot r-dominate v due to pivot-based sorting (in decreasing order).

In addition, we adopt a directed acyclic graph (DAG) [22], [28] to maintain all pair-wise r-dominance relationships between vertices in H_k^t, named *r-dominance graph* (denoted by G_d). Fig. 4(b) illustrates G_d of H_3^9. An arc from vertex v to v' signifies that v r-dominates v'. It is noting that an arc from v_6 or v_2 to v_7 is not needed as the transitivity of r-dominance relationship already implies this. The number of vertices that r-dominate v is called v's *r-dominance count*.

V. Global Search

In this section, we develop a global search algorithm for Problem 2 and discuss its generalization for Problem 1. Before proceeding further, three useful lemmas are given as follows.

Lemma 4: The maximal (k, t)-core, i.e., H_k^t, is an MAC.

Lemma 5: For any MAC, if we delete the smallest-score vertex w.r.t. any weight w in R and the resulting subgraph still has a k-$\widehat{core}$ H containing Q, H is an MAC.

Lemma 6: For any MAC H, if we delete the smallest-score vertex w.r.t. any weight w in R but the resulting subgraph does not have a k-$\widehat{core}$ containing Q, H is a non-contained MAC.

In view of the above lemmas, we can devise an efficient algorithm based on depth-first search (DFS) for our problems.

A. The DFS-based Algorithm

The idea of the DFS-based algorithm is described in detail in Algorithm 1. First, for given k and t, we compute the maximal (k, t)-core, i.e., H_k^t, and build the r-dominance graph G_d. Then, the following procedure is iteratively invoked until the resulting graph in each partition of R does not have a k-$\widehat{core}$ containing Q. The procedure consists of two steps. Let H and G'_d be the resulting subgraph and corresponding r-dominance graph in partition ρ of R (Line 6). The first step is to insert sub-partitions into ρ according to G'_d (Lines 7-9), then find the smallest-score vertex in each sub-partition (Lines 10-11). In Line 9, note that ρ is the root node of a binary tree after being passed in as a parameter, and S is a set of leaf nodes of the binary tree, representing sub-partitions of ρ. This step is essential and will be elaborated in Section V-B. The second step is to delete all the vertices that are definitely excluded in subsequent MACs, which enables H and G'_d to be updated accordingly (Lines 15-20).

Algorithm 1: DFS-based algorithm

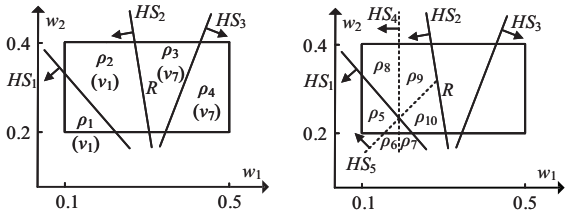
Input: G_r, G_s, Q, k, t, R
Output: The top-j MACs
1 $G'_s \leftarrow$ filter out vertices by RangeQuery(Q, t);
2 $H_k^t \leftarrow$ k-core decomposition on G'_s;
3 $G_d \leftarrow$ build r-dominance graph of H_k^t;
4 Queue $U \leftarrow \emptyset$; Heap $I \leftarrow \emptyset$; U.push(H_k^t, G_d, R, I);
5 **while** $U \neq \emptyset$ **do**
6 $(H, G'_d, \rho, I') \leftarrow U$.pop();
7 $HP \leftarrow$ compute/locate new hyperplanes via leaf nodes of G'_d;
8 **foreach** $hp \in HP$ (if exists) **do**
9 Sub-partitions $S \leftarrow$ **Partition**(ρ, hp);
10 **foreach** $\rho' \in S$ **do**
11 $u \leftarrow$ find the smallest-score vertex (if $\notin Q$);
12 DFS(u, H, G'_d, I'); {// Consider condition (2) in Corollary 1}
13 **if** Corollary 1 holds **then** output top-j MACs of ρ';
14 **else** U.push(H, G'_d, ρ', I');
15 **Procedure** DFS(u, H, G'_d, I)
16 **foreach** $v \in N(u, H)$ **do**
17 Delete edge (u, v) from H;
18 **if** $dg_H(v) < k$ **then**
19 DFS(v, H, G'_d, I);
20 I.push(u); delete u from H and G'_d (with incident edges);

In particular, Algorithm 1 recursively deletes all the vertices violating the structural cohesiveness constraint by a DFS procedure (Lines 16-19) as long as the smallest-score vertex u is found in the sub-partition. Because the degrees of the adjacent vertices of u all reduce by 1 when we delete u. This may cause some neighbors of u to violate the structural cohesiveness constraint and thus cannot be contained in subsequent MACs. Likewise, we also need to verify whether the neighbors of other hops (e.g., 2-hop, 3-hop, etc.) meet the structural cohesiveness constraint. Obviously, the DFS procedure can be used to identify and delete all these vertices.

According to Lemma 6, we have a corollary as follows.

Corollary 1: Given an MAC H, H is a non-contained MAC if the smallest-score vertex u meets one of the following conditions: (1) $u \in Q$; and (2) deleting u will recursively disconnect Q (e.g., $\exists q \in Q$ being deleted) or make the degree of remaining vertices less than k.

Note that we always consider the early termination conditions of Corollary 1 in conjunction with the DFS procedure. Once either is met, it means that vertex u cannot be deleted, even if u is currently the one with the smallest score but H is



(a) Partitioning R (b) Partitioning ρ_1 and ρ_2

Fig. 5. Arrangement and partitions in R

already the non-contained MAC w.r.t. partition ρ' (Line 12). As a result, if Corollary 1 (i.e., Lemma 6) holds, the top-j MACs can be obtained by the union of top vertices in heap I' (totally backtracking $j-1$ times) and the last subgraph H (Line 13). Based on Lemma 4 and 5, we can easily verify that H for each partition ρ recursively obtained in Line 6 is an MAC. Thus, Theorem 1 shows the correctness of Algorithm 1.

Theorem 1: Algorithm 1 correctly finds the top-j MACs.

Proof Sketch: For any partition ρ, as long as its current subgraph H does not hold Corollary 1, ρ will be divided into $|S|$ sub-partitions by $|HP|$ hyperplanes (Line 10 in Algorithm 1), e.g., ρ_i for $1 \leq i \leq |S|$. Here, ρ is discarded when the recursion proceeds to the promising sub-partitions of the local arrangement, i.e., ρ_i. Assume that u is the smallest-score vertex in H w.r.t. ρ_i, the resulting subgraph, denoted by H_i, obtained by invoking DFS procedure (Line 12 in Algorithm 1) can be claimed as the maximal k-$\widehat{core}$ of the subgraph $H \backslash u$ by contradiction, because DFS procedure recursively deletes all the vertices in H whose degrees are smaller than k. Therefore, we have $V_{H_i} \subset V_H$ and $S(H) \leq S(H_i)$ w.r.t. ρ_i for $1 \leq i \leq |S|$. In this way, the non-contained MAC corresponding to each final sub-partition can be found, as well as all the MACs. Thus, we conclude that Algorithm 1 correctly finds the top-j MACs. □

We analyze the complexity of Algorithm 1 in Theorem 2.

Theorem 2: The time complexity and space complexity of Algorithm 1 are bounded by $O(n'^{2d})$ and $O(m'+n'+n'^{2}\cdot d)$ respectively, where n' and m' denote the number of vertices and edges in H_k^t.

Proof: The key factor determining Algorithm 1's time complexity is the construction of arrangements. In the worst case, vertices in H_k^t are r-incomparable to each other, i.e., the complete arrangement of $\frac{n'(n'-1)}{2}$ half-spaces needs to be constructed, in $O(n'^{2d})$ time [29]. The algorithm only needs to store H_k^t, and maintains the heap I' and half-space information related to d, which uses less than $O(m'+n'+n'^{2}\cdot d)$ space complexity even in the worst case. □

B. Arrangement Jointing and Indexing

In Algorithm 1, to find the smallest-score vertex for any weight vector in partition ρ, we consider the vertices of G'_d in a bottom-up manner. In other words, leaf vertices of the r-dominance graph will be preferred (Line 7). The reason is that if a vertex is deleted either because it does not satisfy the structural cohesiveness constraint (already considered in the DFS procedure), or because it is the smallest-score vertex, but before this, all vertices it r-dominates should be deleted first.

Algorithm 2: Partition

Input: Node ρ, hyperplane hp
Output: Leaf nodes of the binary tree of half-space arrangements
1 **if** $\rho \cap hp^- = \emptyset$ **then** ρ is covered by hp^+;
2 **else if** $\rho \cap hp^+ = \emptyset$ **then** ρ is covered by hp^-;
3 **else**
4 **if** ρ's child is *NULL* **then**
5 Insert $\rho.left \leftarrow \rho \cap hp^-$ and $\rho.right \leftarrow \rho \cap hp^+$;
6 **else**
7 **Partition**($\rho.left, hp$);
8 **Partition**($\rho.right, hp$);

The verification of a leaf vertex u in G'_d entails partitioning ρ by half-spaces $HS_i : S(u') \geq S(u)$, each corresponding to one of the remaining leaf vertex u'. Formally, an arrangement bounded by R is defined by the supporting hyperplanes of these half-spaces, where each cell (i.e., sub-partition) is located in a set of half-spaces. The leaf vertices corresponding to these half-spaces are precisely those with scores higher than u if w falls in that cell, which means u is the smallest-score vertex.

Consider G_d in Fig. 4(b). Initially, the leaf vertices are v_7, v_5 and v_1 ($G'_d = G_d, I' = \emptyset$, Line 6 in Algorithm 1). Their respective half-spaces $HS_1 : S(v_7) \geq S(v_5)$, $HS_2 : S(v_7) \geq S(v_1)$ and $HS_3 : S(v_1) \geq S(v_5)$ are inserted into the arrangement, as shown in Fig. 5(a). The vertex in brackets for each partition indicates the smallest-score vertex for any weight vector w in that partition. For partitions ρ_3 and ρ_4 on the right, vertex v_1 will also be deleted by the DFS procedure due to the deletion of v_7, after which v_1 and v_7 are both pushed into the heap I' w.r.t. the partitions (representing the vertices ignored). Thus, the resulting subgraph H induced by $\{v_2, \ldots, v_6\}$ is the corresponding non-contained MAC since discarding any vertex will no longer satisfy the structural cohesiveness constraint. By backtracking the top vertices in I' once (i.e., v_1 and v_7), we can easily obtain the second-ranked MAC induced by $\{v_1, \ldots, v_7\}$ in ρ_3 and ρ_4 (refer to R_3 in Fig. 2(b)).

When Corollary 1 does not hold, sub-partition ρ' will be pushed into queue U to compute the non-contained MAC in depth. At this time, vertices ignored will be discarded to update resulting subgraph H and corresponding G'_d, so that new leaf vertex can be designated in the next round of verification and the half-spaces against other leaf vertices are inserted into the local arrangement. Consider ρ_1 in Fig. 5(a), we refer to v_4 as the new leaf vertex after v_1 is deleted, i.e., H and G'_d are induced by $\{v_2, \ldots, v_7\}$ and $I' = \{v_1\}$. Then, new half-spaces $HS_4 : v_7 \geq v_4$ and $HS_5 : v_5 \geq v_4$ are inserted into a newly initialized local arrangement against partition ρ_1, where three sub-partitions ρ_5, ρ_6 and ρ_7 are produced, as shown in Fig. 5(b). As v_4 and v_5 are pushed into I' together, the non-contained MAC induced by $\{v_2, v_3, v_6, v_7\}$ is returned for each sub-partition. Likewise, same operation applies to partition ρ_2. Eventually, the solution in Example 3 is obtained. Note that we can directly locate HS_4 and HS_5 for ρ_2 since no new half-space needs to be computed due to the same leaf vertices (in G'_d) as ρ_1. This drastically reduces repetitive computation in half-spaces, each of which is computed only once if necessary.

Algorithm 3: Local Search Framework

Input: G_r, G_s, Q, k, t, R
Output: The non-contained MACs
$G'_s \leftarrow$ filter out vertices by RangeQuery(Q, t);
$H^t_k \leftarrow k$-core decomposition on G'_s;
$G_d \leftarrow$ build r-dominance graph of H^t_k;
$C \leftarrow$ **Expand**(H^t_k, G_d, Q, k);
Verify(C, G_d, Q, R);

Specifically, for each local arrangement considered, an index is built by a recursive process *Partition* (Algorithm 2). Then, the index is discarded when all relevant half-spaces are inserted, leaving only the hopeful sub-partitions of the local arrangement (if any). Except that, optimization of arrangement indexing and maintenance is the same as described in [13].

VI. Local Search

Although the efficiency of the global search algorithm is considerable for each query, it may need to explore the entire maximal (k, t)-core, especially when query vertices Q are located at the upper layer of the r-dominance graph G_d. In this section, we devise the local search algorithms for Problem 2 and investigate the generalization for Problem 1.

The intuition is that the non-contained MACs for Q are in the vicinity of Q. Thus, the entire H^t_k should be unnecessary to involve during the search. Nonetheless, it is intractable to enumerate all the k-$\widehat{core}$s containing Q, whose number can be exponentially large w.r.t. H^t_k size. Accordingly, we only immerse in finding the communities that are most likely to be candidates for non-contained MACs. Their validity and corresponding partitions in R can be quickly verified by G_d alone. This inspires us to develop a framework of more efficient local search (Algorithm 3). Specifically, *Expand* procedure finds candidates (i.e., C) by selecting the most promising vertex as we explore in the neighborhood of Q, and stops when each target community forms a k-$\widehat{core}$. *Verify* procedure provides guarantee of identifying all valid non-contained MACs w.r.t. R from C. Note that the time complexity of Algorithm 3 is bounded by $O(|C|{\cdot}s^d))$ (see Theorem 3 and 4), which is much lower than that of Algorithm 1 ($|C|$ and s are typically very small in practice). As verified in our experiments, local search is at least one order of magnitude faster than global search, and all non-contained MACs will be expanded by our candidate generation strategies in most cases.

In this way, two problems arise: (1) how do we guarantee that the target community can be a candidate for non-contained MACs (at least form a k-$\widehat{core}$ containing Q); and (2) how do we know whether the k-$\widehat{core}$ is a valid non-contained MAC w.r.t. R? The former poses a great challenge of determining which vertex to choose and when to terminate expansion, and the latter requires an in-depth study of the characteristics of non-contained MACs. In the following, we present lemmas and algorithms for the local search strategy.

A. Candidate Generation

To be a candidate for non-contained MACs, the current community must be at least a k-$\widehat{core}$ containing Q. It would be nice if the structural cohesiveness metric (i.e., k-core) is "monotonic", which means that the larger the community, the smaller its minimum degree. So once the minimum degree drops below the given coreness threshold k, we can stop the search. Unfortunately, such a metric is not monotonic. Thus, the greatest challenge is to overcome non-monotonicity first, which motivates us to conduct community search only by exploring local neighborhood of Q.

Algorithm 4: Expand

Input: H^t_k, G_d, Q, k
Output: The candidate set C
Queue $U \leftarrow \emptyset$; $V_H \leftarrow Q$; {// or $\forall v \in N(Q, H^t_k), V_H \leftarrow Q \cup \{v\}$}
foreach unvisited $v \in N(V_H, H^t_k)$ **do**
  U.push(v);
while $U \neq \emptyset$ **do**
  $u \leftarrow U$.pop(); $V_H \leftarrow V_H \cup \{u\}$;
  if $\delta(H) \geq k$ **then** output $C \leftarrow C \cup H$;
  foreach unvisited $v' \in N(u, H^t_k)$ **do**
    U.push(v');

Now for the minimum degree of a subgraph H, denoted by $\delta(H)$, we make an in-depth analysis of its monotonicity. Consider the exploration starting from the query vertices, i.e., H_0 induced by Q. We add a vertex from H^t_k at each step until a k-$\widehat{core}$ H is obtained, assuming that $v_1, v_2, \ldots, v_e$ is a vertex sequence it adds. So let H_i be induced by $Q \cup \{v_1, \ldots, v_i\}$. In general, $\delta(H)$ is a non-monotonic function of H. More formally, $\delta(H_{i+1})$ is unnecessarily greater than $\delta(H_i)$. However, the order of vertices added to V_H determines the monotonicity of $\delta(H)$. Interestingly, for any Q with $\delta(H_0) = 0$, we can always find a sequence of added vertices such that $\delta(H_i)$ is a non-decreasing function of i.

Lemma 7: For any query vertices $Q \subseteq V_H$ with $\delta(H_0) = 0$ in graph H^t_k, there always exists an added vertex sequence $v_1, v_2, \ldots, v_e$ of H starting with Q such that $\forall 0 \leq i \leq e, \delta(H_i) \leq \delta(H_{i+1})$.

Proof Sketch: It is consistent with proving that vertices can be removed one by one from V_H until Q, just ensuring that each removal of v does not increase the minimal degree of the remaining vertices. Otherwise, it occurs only when v is currently one of the vertices with the minimal degree. The reason is that removing a vertex with non-minimal degree will only preserve or decrease the minimal degree. □

Lemma 7 implies that there always exists an exploration order that leads monotonically to a community of k-$\widehat{core}$ containing Q. This can be generated by a sequence of vertices added to V_H starting from Q so that $\delta(H_i) \leq \delta(H_{i+1})$ for each i. Note that the existence of such an order is a necessary but insufficient condition for finding a valid community. To illustrate the insufficiency, consider $Q = \{v_9\}$ and $k = 2$ in Fig. 1(a). Any vertex sequence starting with v_9, v_{14} cannot yield a valid solution, yet $\delta(H_1)$ is greater than $\delta(H_0)$.

In *Expand* procedure (Algorithm 4), we explore from the vicinity of Q by BFS and generate candidate set C. To converge the current community towards a candidate for non-contained MAC, we develop two intelligent candidate selection strategies according to Lemma 3, 6 and 7. The idea of improving candidate generation is to use priority queues such that

the most promising vertex can be selected to rapidly generate a candidate. From the perspective of structural cohesiveness, the priority of a vertex v can be defined as $f_1(v) = \delta(H') - \delta(H)$ or $f_2(v) = dg_{H'}(v)$, where $V'_H = V_H \cup \{v\}$. $f_1(v)$ emphasizes the improvement of minimum degree for the next step, with $f_1(v) = 1$ or 0 for any v; $f_2(v)$ produces the fastest increase in average degree of H so that the minimum degree will increase with H's density growth. From the perspective of community score, the priority of v can be defined as $f_3(v) = \zeta - l(v)$, where ζ is a constant (maximum priority in G_d) and $l(v)$ denotes the layer of v in G_d. $f_3(v)$ drives community score higher by adding a vertex that r-dominates as many vertices as possible. To sum up, the priority $f(v)$ is defined as

$$f(v) = \lambda \cdot f_2(v) + f_3(v), \tag{3}$$

where λ is a trade-off against ζ, or

$$f(v) = \zeta \cdot f_1(v) + f_3(v). \tag{4}$$

Theorem 3: The time complexity of Algorithm 4 by Eq. 3 and Eq. 4 is $O(\overline{n}+\overline{m})$ and $O(\overline{n}+\overline{m}log\overline{n})$ respectively, where $\overline{n}$ and $\overline{m}$ denote the number of vertices and edges in C. The space complexity is $O(\overline{n}+\overline{m})$.

B. Verification

The determinant of global search is that computing the local arrangement of all half-spaces HS_i among leaf vertices is a relatively expensive process (in $O(i^d)$ time [29], where i is the number of leaf vertices). Instead, in *Verify* procedure (Algorithm 5), an empty arrangement in R is initialized, into which half-spaces w.r.t. a carefully selected and therefore very small subset of vertices (i.e., *competitors* below) are inserted, expecting to securely confirm or disqualify candidate H without considering all other vertices. But before this, we first give a corollary to filter out unpromising candidates from C. Note that G_e represents the r-dominance graph corresponding to H, which is a subgraph of G_d induced by V_H, denoted by $G_d[V_H]$; and G_c represents the rest of G_d, denoted by $G_d[V_{G_d} \backslash V_H]$.

Corollary 2: A community H can be discarded if one of the following conditions is met: *(1)* $\forall v \in V_{G_d} \backslash V_H$, v is a non-leaf vertex in G_d; and *(2)* $\exists v \in V_{G_d} \backslash V_H, v' \in V_H$ and $v \succ v'$, v cannot be recursively deleted by deleting vertices in $V_{G_d} \backslash V_H$.

Proof Sketch: Vertices in $V_{G_d} \backslash V_H$ must be deleted if H holds. □

In other words, either there exists a non-cross-layer arc in G_c between a non-leaf vertex v and a leaf vertex, or v can be recursively deleted by the DFS procedure. We refer to the H that holds Corollary 2 as a *promising community*.

Lemma 8: For any promising H, v is regarded as an anchor if H still forms a k-$\widehat{core}$ after a (non-Q) leaf vertex v in G_e is deleted.

Now we discuss the verification process by half-space insertion, in which it may further benefit from the r-dominance relationships stored in G_d as follows.

Lemma 9: Consider u, u', and their half-space HS_i inserted into the arrangement. Assume that u'' is a vertex r-dominated by u', and ρ is a partition in the arrangement not covered by HS_i. Thus u is guaranteed to r-dominate u'' in partition ρ.

Proof: First, from the definition of half-space HS_i, $S(u') < S(u)$ holds anywhere outside HS_i. Second, $S(u'') \leq S(u')$ holds anywhere inside R by the definition of r-dominance. Thus, we derive that for each partition ρ of the arrangement outside HS_i, $S(u'') < S(u)$, that is, u r-dominates u'' in ρ. □

Based on Lemma 9, competitors chosen are the vertices in the bottom layer (leaf vertices) of G_e and in the top layer (with r-dominance count 0) of G_c, denoted by $l_b(G_e)$ and $l_t(G_c)$ respectively. The fundamental is that intuitively such competitors are the strongest, which are most likely to assist in disqualifying invalid candidates w.r.t. R in turn.

Corollary 3: For any promising H, it is a valid non-contained MAC if a partition exists in R such that all vertices in $l_b(G_e)$ score higher than those in $l_t(G_c)$, and the corresponding conditions are met:

1. If Lemma 8 holds, additionally, all anchors need to score higher than other leaf vertices in G_e.
2. $\exists v \in l_t(G_c)$, if v can be recursively deleted by the DFS procedure starting from $l_b(G_c)$ (namely, v is bound), then $l_t(G_c)$ is updated where v is ignored and replaced by the vertices of its next layer in G_c.
3. $\exists v, v' \in l_t(G_c)$, if v and v' are bound to each other, then vertices in $l_b(G_e)$ only need to score higher than v or v'.

Proof: According to Corollary 2, all vertices in $V_{G_d} \backslash V_H$ have to be deleted when H is a promising community. In other words, vertices in $V_{G_d} \backslash V_H$ are either recursively deleted by deleting those in the lower layer of G_c, or deleted individually due to their lower scores. First, we consider the latter case. As a sufficient and necessary condition, these vertices just need to score lower than those in $l_b(G_e)$; that is, only vertices in $l_b(G_e)$ score higher than those in $l_t(G_c)$. Then, we consider the former case. That is, if condition (2) holds, it means that the restriction on half-spaces between the competitors can be relaxed by the newly updated vertices in $l_t(G_c)$, since such a vertex v will be deleted anyway. Similarly, if condition (3) holds, the restriction on half-spaces between the competitors can also be relaxed through such vertices, because deletion of one will also lead to deletion of the other. On this basis, H becomes a non-contained MAC when Lemma 8 does not hold; otherwise, condition (1) has to be satisfied. This is because such an anchor can still be deleted as it is a non-Q leaf vertex in G_e, i.e., possibly the current smallest-score vertex in H.

Algorithm 5: Verify

Input: C, G_d, Q, R
Output: The non-contained MACs and corresponding partitions

```
foreach H ∈ C do
    G_e ← G_d[V_H]; G_c ← G_d[V_{G_d} \ V_H];
    if Corollary 2 holds then continue;
    HP ← compute hyperplanes via G_e and G_c (Corollary 3);
    foreach hp ∈ HP do
        Sub-partitions S ← Partition(R, hp);
    foreach ρ ∈ S do
        if Corollary 3 holds then output H and ρ;
```

TABLE II
DATASETS (K=10^3 AND M=10^6)

Dataset	Vertices	Edges	dg_{avg}	dg_{max}	k_{max}
San Francisco (SF)	175K	223K	2.55	8	-
Florida (FL)	1.1M	1.4M	2.53	12	-
Slashdot	79K	0.5M	13	2,507	85
Delicious	536K	1.4M	5	3,216	34
Lastfm	1.2M	4.5M	7	5,150	71
Flixster	2.5M	7.9M	6	1,474	69
Yelp	3.6M	9.0M	5	10,433	129

Putting it all together, we ensure the correctness of the partition corresponding to the non-contained MAC H (if any in R). □

To illustrate Algorithm 5, let us reconsider Example 2. Assume that by Algorithm 4 we have three promising communities H_1, H_3 and H_4, where $V_{H_1} = \{v_2, v_3, v_6, v_7\}$, $V_{H_3} = \{v_2, \ldots, v_6\}$ and $V_{H_4} = \{v_1, v_2, v_3, v_6, v_7\}$. For H_1, $l_b(G_e) = \{v_7\}$ and $l_t(G_c) = \{v_4, v_5\}$. As v_4 and v_5 are bound to each other in H_3^9 (condition (3) met), we only insert HS_1 and HS_4 into R in Fig. 5(b), and choose the partitions covered by either of them. As a result, H_1 is a valid non-contained MAC for any weight vector of R_1 in Fig. 2(b). Similarly, H_3 is also valid w.r.t. $R_2 \cup R_3$ (condition (2) met) but H_4 is invalid (condition (1) met) as its partition is outside R.

Theorem 4: The time complexity of Algorithm 5 is $O(|C| \cdot (n' + m') + c \cdot s^d)$, where $|C|$ and c denote the number of candidates and the number of promising communities in C respectively, and s is the product of $|l_b(G_e)|$ and $|l_t(G_c)|$. The space complexity is bounded by $O(c \cdot s \cdot d + \overline{n} + \overline{m} + n' + m')$.

Finally, we can simply generalize local search for Problem 1. As the non-contained MAC H with corresponding partition ρ is known, we insert sub-partitions into ρ according to G_c (in a up-bottom manner) and add the highest-score vertex to H. As long as the current H contains an MAC, it will be output. The process terminates until all top-j MACs in ρ are acquired. Consider H_1 and its G_c, half-spaces among vertices in $l_t(G_c)$ (i.e., HS_5) are inserted first into R_1. Then v_5 is added to V_{H_1} for ρ_5 and ρ_8 shown in Fig. 5(b). As v_4 r-dominates v_1, we have the second MAC induced by $\{v_2, \ldots, v_7\}$ in ρ_5 and ρ_8. The same applies to ρ_6 and ρ_7.

VII. EXPERIMENTS

Comprehensive experiments are conducted to evaluate the proposed model and four algorithms, named GS−T, GS−NC, LS−T and LS−NC respectively. GS−T and GS−NC (resp. LS−T and LS−NC) are global search algorithms (resp. local search algorithms) used to compute the top-j MACs and the non-contained MACs, respectively. Note that either of the two candidate selection strategies in Section VI-A can be adopted in LS−T or LS−NC, and we just give the results by using Eq. 3 with $\zeta = 100$ and $\lambda = 10$ (results by using Eq. 4 are similar and omitted to save space). All algorithms were implemented in C++, and all experiments were conducted on an Ubuntu server with 2GHz Intel Xeon E7-4820 CPU and 1TB memory.

Datasets. We use five real-world social networks[2,3] and two road networks (SF[4]/FL[5]) in our experiments. Table II summarizes the statistics of datasets, of which dg_{avg}, dg_{max} and k_{max} denote the average degree, the maximal degree and the maximal core number, respectively. Note that numerical attributes are not contained in the first four original social networks[2], for which we employ a widely used method in [21] to generate three different types of numerical attributes, i.e., *independence*, *correlation* and *anti-correlation*. Due to space limit, we report the results obtained from datasets with independent and real attributes. In addition, we map each user v to a spatial point p in the road network that matches the scale of his/her social network as follows: we project SF/FL into range $[0, 1]$ in each dimension and generate normalized $L(v)$ by drawing from a list of recent check-ins. Assume that p is the current location of v if it has the smallest Euclidean distance to $L(v)$ in the projection space.

TABLE III
PARAMETERS

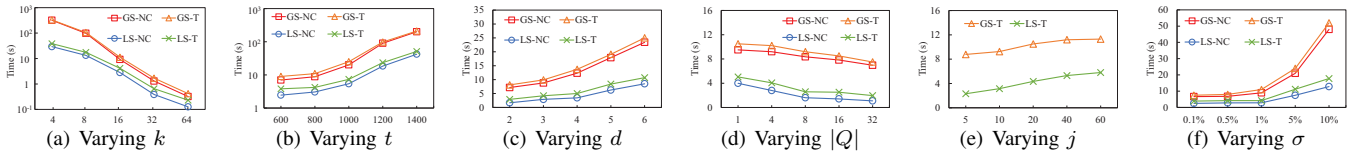
Parameter	Tested values
k	4, 8, **16**, 32, 64
t (SF/FL)	600/800, **800/1000**, 1000/1200, 1200/1400, 1400/1600
d	2, **3**, 4, 5, 6
$\|Q\|$	1, **4**, 8, 16, 32
j	5, 10, **20**, 40, 60
σ	0.1%, 0.5%, **1%**, 5%, 10%

Parameters. We vary 6 parameters: structural cohesiveness k, query distance t, dimensionality d, number of query users $|Q|$, number j of top MACs, and percentage σ of axis length (i.e., side-length of R). Table III shows the range of parameters and their default values (in bold). For each value of $|Q|$, we randomly select 100 sets of query vertices, that satisfy t and can ensure the existence of the maximal (k, t)-core, from the k-core of each social network. In each experiment, only one parameter varies and the rest remains at the default. Every reported measurement is the average of $1,000$ MAC searches for ten randomly generated axis-parallel hypercubes R in the preference domain.

A. Performance Evaluation

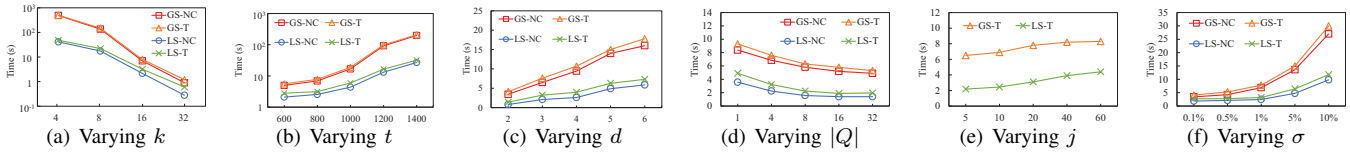
Exp-1: Varying k. We evaluate the query processing time of all algorithms and the number of vertices of H_k^t by varying k. In each road-social network, we can see that local search performs better than global search, but the advantage becomes less obvious when k increases. In the best case, e.g., $k = 4$, LS−T and LS−NC are more than one order of magnitude faster than GS−T and GS−NC in Fig. 7(a) and 9(a). Only when $k = 64$, global search is comparable to local search since H_k^t size shrinks when k is large (see Fig. 11(c)), resulting in a reduction in the time complexity of global search and in the number of promising vertices involved in local search. Although Delicious is larger than Slashdot, but algorithms run faster at $k = 16$ and $k = 32$ since H_k^t contains fewer vertices (as $k_{max} = 34$ in Table II). Note that when k increases from 4 to 8, local search is merely more than twice as fast, e.g., LS−NC

[2]http://networkrepository.com
[3]https://www.yelp.com/dataset
[4]https://www.cs.utah.edu/~lifeifei/SpatialDataset.htm
[5]http://www.dis.uniroma1.it/challenge9/index.shtml



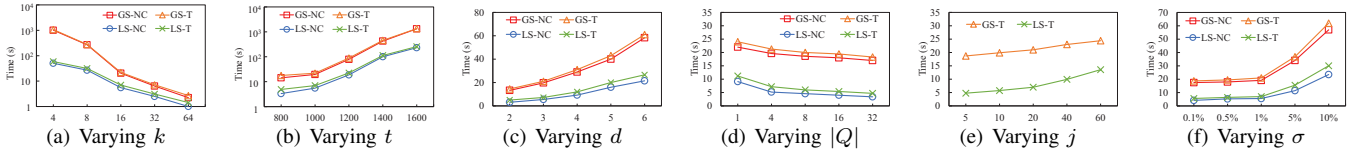
(a) Varying k (b) Varying t (c) Varying d (d) Varying $|Q|$ (e) Varying j (f) Varying σ

Fig. 6. Efficiency and scalability of proposed algorithms in SF+Slashdot with independent attributes.



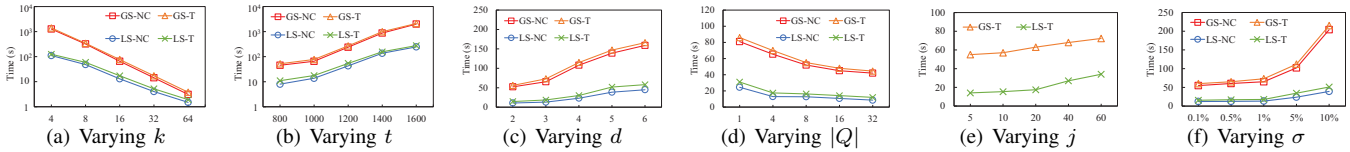
(a) Varying k (b) Varying t (c) Varying d (d) Varying $|Q|$ (e) Varying j (f) Varying σ

Fig. 7. Efficiency and scalability of proposed algorithms in SF+Delicious with independent attributes.



(a) Varying k (b) Varying t (c) Varying d (d) Varying $|Q|$ (e) Varying j (f) Varying σ

Fig. 8. Efficiency and scalability of proposed algorithms in FL+Lastfm with independent attributes.



(a) Varying k (b) Varying t (c) Varying d (d) Varying $|Q|$ (e) Varying j (f) Varying σ

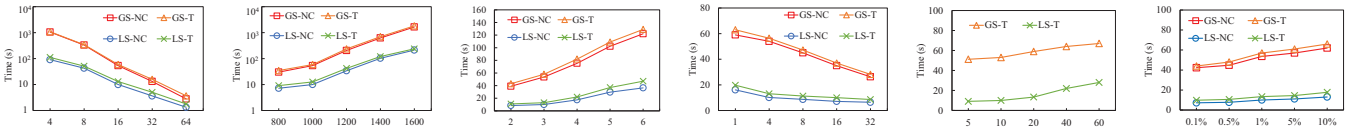
Fig. 9. Efficiency and scalability of proposed algorithms in FL+Flixster with independent attributes.

takes 41s and 17s respectively in Fig. 7(a). This is because when k is relatively small, candidate selection strategies tend to find more smaller candidates. However, algorithms run faster in Yelp than in Flixster, yet its H_k^t size is larger. This will be explained in Exp-6. In short, across a wide range of k local search is consistently better than global search.

Exp-2: Varying t. We evaluate the query processing time of all algorithms by varying t. For each road-social network, local search outperforms global search significantly, with advantage becoming more obvious as t increases. For example, in Fig. 9(b), `GS-NC` takes $1,992$s while `LS-NC` takes 245s for $t = 1600$. Note that the results are obtained in the case of $k = 16$, thus generally `LS-T` and `LS-NC` are almost one order of magnitude faster than `GS-T` and `GS-NC` in terms of t. Because when t is large, more users are retained via range query accelerated by G-tree [24] or G*-tree [25]. This favors local search radiating outward from Q, equivalent to increasing the expansion radius.

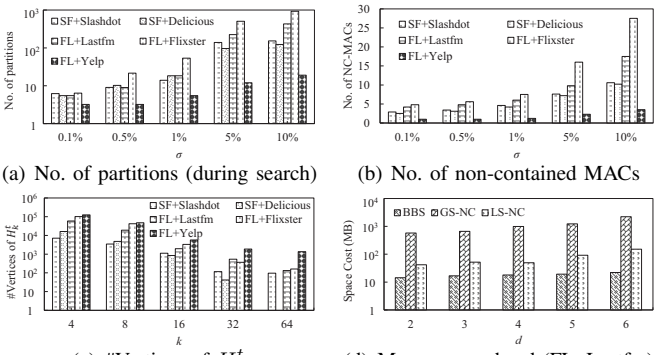
Exp-3: Varying d. By varying d we study the query processing time and the memory overhead of all algorithms, and comparison of different methods. The toughness of MAC search rises with d due to its computational geometric nature. Nonetheless, all four algorithms offer practical processing time, taking respectively 159s and 45s for $d = 6$ in Fig. 9(c). Furthermore, Fig. 11(d) shows the memory overhead of `GS-NC`/`LS-NC` and *BBS* process (X indexed and G_d built). When d increases, dimension of R-tree increases but memory overhead changes not much due to unchanged G_d size; local search is very lightweight against global search. For example, `GS-NC` takes 664MB and $2,219$MB while `LS-NC` takes only 52MB and 152MB for $d=3$ and $d=6$, respectively. The results confirm both the theoretical analysis and the claims on arrangement indexing in Section V-B. In addition, Fig. 13 and 14 show the comparison with methods in [4] and [8], where `Influ` (resp. `Influ+`) is the DFS-based (resp. ICP-index based) algorithm, and `Sky` (resp. `Sky+`) is the basic (resp. space-partition) algorithm. We implement `Influ` and `Influ+` by varying k instead of d since they can only capture 1-dimensional attribute. For a fair comparison, 100 weight vectors that fall anywhere in R are randomly selected to respectively calculate the weighted sum of d (at the default) numerical attributes as vertex influence (i.e., score), and the average processing time is reported in Fig. 13(b) and 14(b). Since no r-dominance graph needs to be maintained and no half-space has to be computed nor inserted, `Influ` and `Influ+` are superior to `GS-NC` and `LS-NC` in terms of processing time, while `Sky` and `Sky+` are generally the most expensive due to their time complexity. On the other hand, in terms of d, `Sky` and `Sky+` are much costlier than ours and intractable when d is relatively large, e.g., $d \geq 3$ and $d \geq 5$ respectively in Fig. 14(c). Here, "Inf" means processing time exceeds $10,000$s. Therefore, our model and algorithms are tractable and scalable to handle real-world applications comprehensively and flexibly.

Exp-4: Varying $|Q|$. We evaluate the query processing time of all algorithms and the ratio of non-contained MACs (NC-MACs) found by `LS-NC` to `GS-NC` by varying $|Q|$. All processing time almost monotonically decreases with the growth of $|Q|$ since it accelerates the convergence of both global and local search. The reason is that increasing $|Q|$ means more vertices cannot be deleted in global search and selecting fewer vertices may find a candidate in local search. Note that in Fig. 7(d), as $k = 16$, algorithms are not much faster at $Q = 32$ than $Q = 16$. In fact, we also find that global search terminates early if the generated query vertices Q are



(a) Varying k (b) Varying t (c) Varying d (d) Varying $|Q|$ (e) Varying j (f) Varying σ

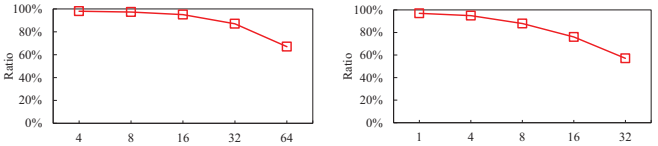
Fig. 10. Efficiency and scalability of proposed algorithms in FL+Yelp with independent attributes.



(a) No. of partitions (during search) (b) No. of non-contained MACs

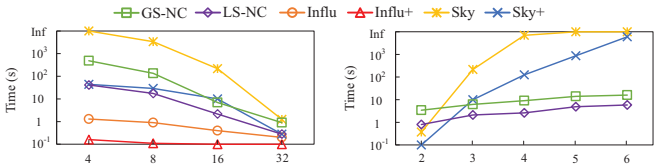
(c) #Vertices of H_k^t (d) Memory overhead (FL+Lastfm)

Fig. 11. Scalability of proposed algorithms.



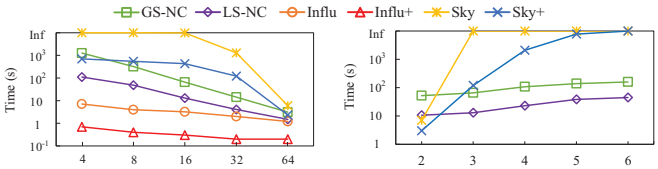
(a) Varying k (b) Varying $|Q|$

Fig. 12. Ratio of NC-MACs found by LS-NC to GS-NC in FL+Lastfm.



(b) Varying k (c) Varying d

Fig. 13. Comparison of different methods in SF+Delicious (independence).



(b) Varying k (c) Varying d

Fig. 14. Comparison of different methods in FL+Flixster (independence).

located in the lower layer of G_d because it is more likely to encounter Q when deleting vertices, while local search is just the opposite. Fig. 12(a) and 12(b) show the ratio against k and $|Q|$, respectively. We can see that both the ratios decrease with the growth of k and Q, but it is still satisfactory. The reason is that the number of community candidates expanded by the *Expand* procedure tends to decrease with the increase of k or $|Q|$, but the *Verify* procedure can always ensure the correctness of the corresponding partitions in R of any (promising) community. On the other hand, this proves that local search is very useful for applications expecting only part of MACs. Note that, in practice, $|Q|$ is usually not very large and the ratio reaches 95% at default $|Q|$, which confirms that local search can find all non-contained MACs in most cases.

Exp-5: Varying j. We examine the query processing time of `GS-T` and `LS-T` by varying j. The curve of `GS-T` is rising slowly with increasing j since the top-j MACs can be directly obtained after executing global search. But for `LS-T`, after obtaining the non-contained MAC, its corresponding cell has to be divided again to find the top-j MACs, resulting in an increase in processing time.

Exp-6: Varying σ. We evaluate the query processing time of all algorithms, and the number of partitions and non-contained MACs by varying σ (determining the size of region R). As anticipated, a larger R means a larger output, thereby more computations needed. In Fig. 11(a), we can see that the growth of σ will lead to a significant increase in the number of partitions in R, which also explains the increase in processing time of all algorithms. Fig. 11(b) records the relationship between the number of non-contained MACs obtained by `GS-NC` and σ. Similarly, as the number of partitions increases, the diversity of non-contained MACs w.r.t. R will also increase. Note that both bars of Yelp in Fig. 11(a) and 11(b) are much shorter than those of Flixster while its H_k^t size is larger in Fig. 11(c). Since attributes not only in Yelp but in real world are usually correlated or more, fewer (even unique) branches in DAG result in less processing time, that is, less half-space computation and insertion.

B. Case Study

NA+Aminer. We apply the road network of North America[4] (NA) and the Aminer (aminer.org) for the first case study. The Aminer is a scientific collaboration network that incorporates authors in DB, DM, IR, and ML fields, comprised of $109,931$ vertices and $300,000$ edges. For each author we crawl four numerical attributes: *h-index*, *#publications*, *activeness*, and *diverseness*. To evaluate the effectiveness of MAC model in real world, we map each author to the location in NA according to its affiliation and use $Q = \{$"Jiawei Han", "Jian Pei", "Philip S. Yu", "Xifeng Yan"$\}$, who are renowned scientists in DM (i.e., relatively high scores), as query vertices. After setting $k = 5, j = 2$ and $R = [0.1, 0.3] \times [0.3, 0.5] \times [0.05, 0.1]$ (with t large enough), Fig. 15(a-d) show the top-2 MACs anywhere in R. Furthermore, we compare MAC with different models. For `InfC` [4], Fig. 15(f, g) report the results involving Q, respectively taking *#publications* only and weighted sum (by $w = (0.3, 0.4, 0.1)$) as influence. In fact, `InfC` either cannot capture the characteristics of all attributes, or must be covered by a non-contained MAC (NC-MAC) if $w \in R$ (e.g., Fig. 15(c)). For `SkyC` [8], there are two results, one is the same as NC-MAC in Fig. 15(a), while the other shown in Fig. 15(e) only contains partial Q and is covered by NC-MAC in Fig. 15(c). In effect, we find that `SkyC` is always contained in NC-MACs due to no query vertices and its skyline property. For `ATC` [7], Fig. 15(h) reports the $(6, 2)$-truss w.r.t.

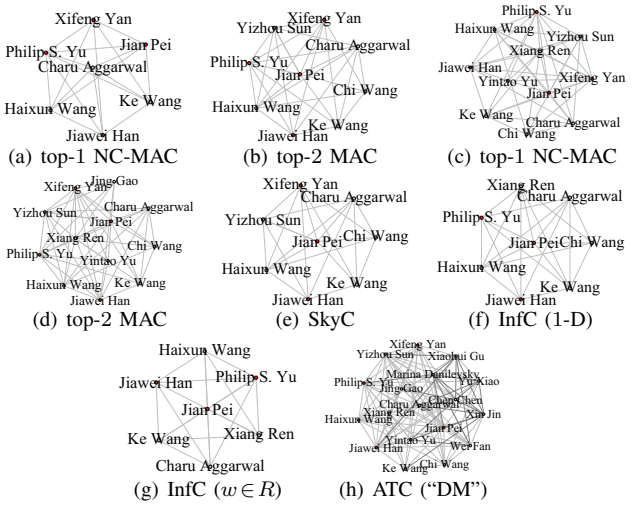


Fig. 15. Case study of Aminer+NA: results for $k = 5$.

Q and keyword "DM" as a $(k+1)$-truss is a k-core. Although communication cost is low (i.e., 2), its size is still too large since it only considers maximum inclusion of keywords but ignores attributes. Therefore, the MAC model is very effective, comprehensive and flexible for applications.

SF+Yelp. We apply the SF and the Yelp[3] for the second case study. In addition to profile information (e.g., ID, first name, etc.), users in Yelp also have real attribute data, such as average rating of all reviews, #reviews written, #hot compliments and so on. Note that the Yelp is more like an LBSN composed of many relatively small ego-like networks, in which ego-like users usually have more fans or followers, write more reviews and receive more compliments. That is, these ego-like users are highly active, showing high attribute values in each dimension; while ordinary users have very low attribute values in all dimensions (e.g., only browsing without posting). In fact, we find that in real world attributes are usually correlated or more, e.g., most users in Yelp have an attribute value of 0. Thus, the number of branches in DAG will be extremely small or even unique, resulting in less half-space computation/insertion and fewer partitions in R. We map each user to the location in SF according to check-ins and use $Q = \{$"Emi", "Phil", "Dani", "Michelle"$\}$, who are relatively active, as query vertices. To discover a group of people who are more concerned and popular, #hot compliments, #more compliments and #photo compliments are used as three numerical attributes here. By setting $k = 6, t = 300, j = 3$ and $R = [0.4, 0.5] \times [0.1, 0.2]$, Fig. 16 shows the top-3 MACs in R. This fully illustrates that in the real world, the diversity of (non-contained) MACs and the number of corresponding partitions w.r.t. R are very small and user-friendly.

VIII. Discussion

In this section, we briefly discuss the conditions for reducing the computations by possibly reusing previous results (namely, subgraph H) when a user may issue multiple MAC searches that share most of identical parameters. Totally, there are three cases as follows. (1) Only R varies to R'. By checking the overlapping part of these two hypercubes, we can keep the previous results in this part and only compute the results for the rest in R'. In particular, the corresponding results can be directly output if the entire R' is inside R. (2) Only k increases or t decreases. We can directly update the previous H_k^t and corresponding G_d by deleting the vertices that do not meet the constraints of structural cohesiveness and communication cost. For the global search, all previously computed hyperplanes (Line 7 in Algorithm 1) can be preserved and may be used in the new search. Specifically, if t decreases and these vertices are all located lower (resp. higher) than the vertices of previous result (namely, V_H) in G_d, then H remains (resp. delete them from H and invoke Algorithm 1); if k increases, these vertices are all located higher than V_H and H still forms a k-$\widehat{core}$ after deleting them (if any), then continue to invoke Algorithm 1 on H. For the local search, we could continue to expand on H until new k is met; or simply delete the vertices that violate new t from H then continue to expand until k is met. (3) Only k decreases or t increases. That is, more vertices could be added into the previous H_k^t and corresponding G_d. For the global search, if k decreases and these newly added vertices are all located lower (resp. higher) than V_H in new G_d, then invoke Algorithm 1 on H (resp. add them to H and invoke Algorithm 1). If t increases, and these newly added vertices are all located lower than V_H in new G_d, then H remains; or these newly added vertices are all located higher than V_H in new G_d and H still forms a k-$\widehat{core}$ after adding (part of) them, then output new H.

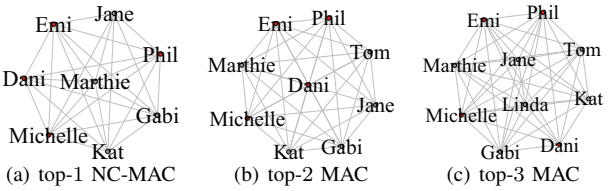


Fig. 16. Case study of Yelp+SF: results for $k = 6$.

IX. Related Work

Community model and search. A large number of community models have been proposed such as k-core [2], [12], k-truss [3], maximal clique [30], quasi-clique [15], maximal k-edge connected subgraph [31]–[33], locally densest subgraph [34], query-biased density [35], etc. All these models consider only graph structural information but ignore numerical/textual attributes associated with vertices. To discover cohesive subgraphs containing the query vertices, a community search problem (CSP) was studied to find the maximal connected k-core in social networks [1], for which [2] proposed a more efficient local search algorithm. Recently, [7] introduced the CSP of small-diameter k-truss with similar query attributes. [5] and [6] developed the CSP of k-core with textual attributes and smallest minimum covering circle, respectively. [4] proposed an influential community with vertex's influence as one numerical attribute, based on which [8] studied a skyline community for d-dimensional numerical attributes. More recently, [36] studied the CSP of k-truss with distance of at most d for any two vertices. [37] and [38] investigated a cohesive version of CSP that brings all community members

closest to the point-of-interest in road networks. [39] and [17] studied two different CSPs in terms of context with only query keywords but no query vertices. In addition, [40] and [41] made variations on the CSPs in [7] and [36], respectively.

This work differs from all the prior work in the following. (1) Our multi-attributed community (MAC) model is the first one that can incorporate uncertainty of user preferences in the weight vector into d-dimensional numerical attributes and capture spatial cohesiveness between users in road networks. (2) The preference domain is introduced into community modeling for the first time. (3) We study the novel MAC search problems in road-social networks such that our techniques are significantly different from all previous CSP algorithms.

Skyline and its generalization. The r-dominance graph used in our MAC model is relevant to the skyline [21] and more to its extension, the k-skyband [26]. In traditional top-j queries, if two records are inconsistent and one has no smaller value in any dimension [21], then it *dominates* the other. Thus, for a dataset the skyline consists of the records which are not dominated by any other; while the k-skyband comprises those dominated by fewer than k other records [26], indicated as a superset of all records which for any weight vector may occur in the top-j results. As a typical k-skyband computation algorithm, *BBS* [26] adopts a spatial index in the dataset, following the branch-and-bound paradigm [42].

This proves an additional essential difference between [8] and our work from another perspective. Regardless of social and spatial cohesiveness, dominance in [8] between two communities comes down to a series of standard dominance tests on d-dimensional vectors. However, r-dominance tests adopted in our model do not suffice, e.g., two or more non-skyline communities may still collaboratively disqualify a skyline one if they score higher than it at different parts of R, collectively blocking it from being a top-j result anywhere in R.

X. Conclusions

In this paper, we propose a novel community model to discover normative communities suitable for multi-criteria decision making in a road-social network, in which each user is linked with location information and d (≥ 1) numerical attributes. Taking a preference region of d-dimensional data domain as input, the resulting communities identified by our model cannot be r-dominated by other ones as long as the weight vector could fall anywhere in the region. We formalize the multi-attributed community search; distinguish two problem versions; develop solutions for corresponding processing; and using both real-world and synthetic datasets demonstrate the efficiency and scalability of our solutions and the effectiveness of our model.

References

[1] M. Sozio and A. Gionis, "The community-search problem and how to plan a successful cocktail party," in *SIGKDD*, 2010, pp. 939–948.

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

[3] X. Huang, L. V. Lakshmanan, J. X. Yu, and H. Cheng, "Approximate closest community search in networks," *PVLDB*, vol. 9, no. 4, pp. 276–287, 2015.

[4] R. Li, L. Qin, J. X. Yu, and R. Mao, "Influential community search in large networks," *PVLDB*, vol. 8, no. 5, pp. 509–520, 2015.

[5] Y. Fang, R. Cheng, S. Luo, and J. Hu, "Effective community search for large attributed graphs," *PVLDB*, vol. 9, no. 12, pp. 1233–1244, 2016.

[6] Y. Fang, R. Cheng, X. Li, S. Luo, and J. Hu, "Effective community search over large spatial graphs." *PVLDB*, vol. 10, no. 6, pp. 709–720, 2017.

[7] X. Huang and L. V. Lakshmanan, "Attribute-driven community search," *PVLDB*, vol. 10, no. 9, pp. 949–960, 2017.

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

[9] T. Joachims, "Optimizing search engines using clickthrough data," in *SIGKDD*, 2002, pp. 133–142.

[10] B. Jiang, J. Pei, X. Lin, D. W. Cheung, and J. Han, "Mining preferences from superior and inferior examples," in *SIGKDD*, 2008, pp. 390–398.

[11] L. Qian, J. Gao, and H. Jagadish, "Learning user preferences by adaptive pairwise comparison," *PVLDB*, vol. 8, no. 11, pp. 1322–1333, 2015.

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

[13] K. Mouratidis and B. Tang, "Exact processing of uncertain top-k queries in multi-criteria settings," *PVLDB*, vol. 11, no. 8, pp. 866–879, 2018.

[14] V. Batagelj and M. Zaversnik, "An o(m) algorithm for cores decomposition of networks," *CoRR, cs.DS/0310049*, 2003.

[15] W. Cui, Y. Xiao, H. Wang, Y. Lu, and W. Wang, "Online search of overlapping communities," in *SIGMOD*, 2013, pp. 277–288.

[16] F. Zhang, Y. Zhang, L. Qin, W. Zhang, and X. Lin, "When engagement meets similarity: efficient (k, r)-core computation on social networks," *PVLDB*, vol. 10, no. 10, pp. 998–1009, 2017.

[17] Z. Zhang, X. Huang, J. Xu, B. Choi, and Z. Shang, "Keyword-centric community search," in *ICDE*, 2019, pp. 422–433.

[18] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *SIGMOD*, 1984, pp. 47–57.

[19] Y.-C. Chang, L. Bergman, V. Castelli, C.-S. Li, M.-L. Lo, and J. R. Smith, "The onion technique: indexing for linear optimization queries," in *SIGMOD*, 2000, pp. 391–402.

[20] B. Tang, K. Mouratidis, and M. L. Yiu, "Determining the impact regions of competing options in preference space," in *SIGMOD*, 2017, pp. 805–820.

[21] S. Borzsony, D. Kossmann, and K. Stocker, "The skyline operator," in *ICDE*, 2001, pp. 421–430.

[22] J. Liu, L. Xiong, J. Pei, J. Luo, and H. Zhang, "Finding pareto optimal groups: Group-based skyline," *PVLDB*, vol. 8, no. 13, pp. 2086–2097, 2015.

[23] I. F. Ilyas, G. Beskales, and M. A. Soliman, "A survey of top-k query processing techniques in relational database systems," *ACM Comp. Surveys*, vol. 40, no. 4, pp. 1–58, 2008.

[24] R. Zhong, G. Li, K.-L. Tan, L. Zhou, and Z. Gong, "G-tree: An efficient and scalable index for spatial search on road networks," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 8, pp. 2175–2189, 2015.

[25] Z. Li, L. Chen, and Y. Wang, "G*-tree: An efficient spatial index on road networks," in *ICDE*, 2019, pp. 268–279.

[26] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive skyline computation in database systems," *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 41–82, 2005.

[27] M. D. Berg, O. Cheong, M. V. Kreveld, and M. Overmars, *Computational geometry: algorithms and applications*. Springer, 2008.

[28] L. Zou and L. Chen, "Pareto-based dominant graph: An efficient indexing structure to answer top-k queries," *IEEE Trans. Knowl. Data Eng.*, vol. 23, no. 5, pp. 727–741, 2011.

[29] P. K. Agarwal and M. Sharir, "Arrangements and their applications," in *Handbook of computational geometry*. Elsevier, 2000, pp. 49–119.

[30] J. Cheng, Y. Ke, A. W.-C. Fu, J. X. Yu, and L. Zhu, "Finding maximal cliques in massive networks," *ACM Trans. Database Syst.*, vol. 36, no. 4, pp. 21:1–21:34, 2011.

[31] R. Zhou, C. Liu, J. X. Yu, W. Liang, B. Chen, and J. Li, "Finding maximal k-edge-connected subgraphs from a large graph," in *EDBT*, 2012, pp. 480–491.

[32] T. Akiba, Y. Iwata, and Y. Yoshida, "Linear-time enumeration of maximal k-edge-connected subgraphs in large networks by random contraction," in *CIKM*, 2013, pp. 909–918.

[33] L. Chang, J. X. Yu, L. Qin, X. Lin, C. Liu, and W. Liang, "Efficiently computing k-edge connected components via graph decomposition," in *SIGMOD*, 2013, pp. 205–216.

[34] L. Qin, R. Li, L. Chang, and C. Zhang, "Locally densest subgraph discovery," in *SIGKDD*, 2015, pp. 965–974.
[35] Y. Wu, R. Jin, J. Li, and X. Zhang, "Robust local community detection: on free rider effect and its elimination," *PVLDB*, vol. 8, no. 7, pp. 798–809, 2015.
[36] L. Chen, C. Liu, R. Zhou, J. Li, X. Yang, and B. Wang, "Maximum co-located community search in large scale social networks," *PVLDB*, vol. 11, no. 10, pp. 1233–1246, 2018.
[37] F. Guo, Y. Yuan, G. Wang, L. Chen, X. Lian, and Z. Wang, "Cohesive group nearest neighbor queries over road-social networks," in *ICDE*, 2019, pp. 434–445.
[38] F. Guo, Y. Yuan, G. Wang, L. Chen, X. Lian, and Z. Wang, "Cohesive group nearest neighbor queries on road-social networks under multi-criteria," *IEEE Trans. Knowl. Data Eng.*, 2020.
[39] L. Chen, C. Liu, K. Liao, J. Li, and R. Zhou, "Contextual community search over large social networks," in *ICDE*, 2019, pp. 88–99.
[40] Q. Liu, Y. Zhu, M. Zhao, X. Huang, J. Xu, and Y. Gao, "Vac: vertex-centric attributed community search," in *ICDE*, 2020, pp. 937–948.
[41] J. Luo, X. Cao, X. Xie, Q. Qu, Z. Xu, and C. S. Jensen, "Efficient attribute-constrained co-located community search," in *ICDE*, 2020, pp. 1201–1212.
[42] A. H. Land and A. G. Doig, "An automatic method of solving discrete programming problems," *Econometrica*, vol. 28, no. 3, pp. 497–520, 1960.