Technical Report: Discovering Personalized Characteristic Communities in Attributed Graphs

Yudong Niu[1], Yuchen Li[1], Panagiotis Karras[2], Yanhao Wang[3], Zhao Li[4]
[1]*School of Computing and Information Systems, Singapore Management University, Singapore*
[2]*Department of Computer Science, Aarhus University, Aarhus, Denmark*
[3]*School of Data Science and Engineering, East China Normal University, Shanghai, China*
[4]*Zhejiang Lab, Hangzhou, China*
ydniu.2018@phdcs.smu.edu.sg, yuchenli@smu.edu.sg, piekarras@gmail.com, yhwang@dase.ecnu.edu.cn, lzjoey@gmail.com

***Abstract*—What is the widest community in which a person exercises a strong impact? Although extensive attention has been devoted to searching communities containing given individuals, the problem of finding their *unique communities of influence* has barely been examined. In this paper, we study the novel problem of *Characteristic cOmmunity Discovery* (COD) in attributed graphs. Our goal is to identify the largest community, taking into account the query attribute, in which the query node has a significant impact. The key challenge of the COD problem is that it requires evaluating the influence of the query node over a large number of hierarchically structured communities. We first propose a novel compressed COD evaluation approach to accelerate the influence estimation by eliminating redundant computations for overlapping communities. Then, we further devise a *local hierarchical reclustering* method to alleviate the skewness of hierarchical communities generated by global clustering for a specific query attribute. Extensive experiments confirm the effectiveness and efficiency of our solutions to COD: they find better characteristic communities than existing community search methods by several quality measures and achieve up to 25× speed-ups against well-crafted baselines.**

I. Introduction

Graphs are a natural model to represent complex relationships between entities in various domains. Since rich information is becoming widely available for real-world entities, nodes in graphs are often associated with *attributes*. For example, an attributed graph can represent academic collaborations among researchers, each of whom is described by areas of expertise, affiliations, and other information in the profile data. To provide personalized service to users, different methods are proposed to find communities that contain the query nodes while having high structure and attribute cohesiveness [1]–[5]. In addition, considerable effort has also been devoted to incorporating the notion of influence in community search problems [6]–[8], which aims to identify communities with strong influences on all nodes of the graph. However, the problem of discovering a query node's (e.g., a person's or an item's) *characteristic community*, that is, the widest community in which the node has a significant impact as well as a high relevance, has not yet been studied, albeit its prominence in real-world applications.

Specifically, let us consider the domain of community-based social marketing (CBSM), a powerful approach to transforming individual behaviors and shaping opinions within a community context [9]–[11]. A notable application of CBSM is Amazon's initiative, which engaged community promoters to endorse the brand through personal stories. This approach successfully expanded the dialogue around online shopping, with community discussions rising from 168,000 to 220,000 instances and a three-fold increase in Amazon's brand mentions[1]. A characteristic community can be utilized to optimize such CBSM efforts by ensuring that promoters reach not only large communities but also those where their influence is most pronounced. This strategic focus is crucial across various application domains where CBSM is applied, such as the promotion of new brands [12], environmentally friendly lifestyles [11], or academic conferences [13].

Existing attributed community search methods [1]–[5] solely prioritize the community cohesion. In contrast, the characteristic community incorporates promoter influences, optimizing both community cohesiveness and influence receptivity. This dual focus results in communities that are more engaged and influenced, a critical factor for the success of CBSM efforts [14].

The following example serves to highlight the advantages of the characteristic community over the communities discovered by traditional community search methods.

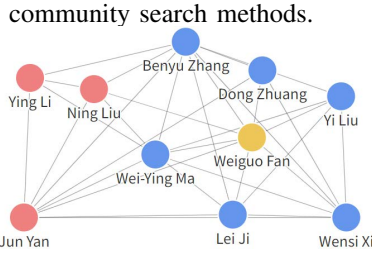


Fig. 1. The community discovered by an attributed community search method ATC [1] (in red) and the characteristic community (in blue) for the query researcher Weiguo Fan (in yellow).

Example 1. *Fig. 1 illustrates the community discovered by an attributed community search method ATC [1] and the characteristic community for a query node Weiguo Fan (in yellow) in the coauthor network extracted from* DBLP. *ATC finds a 3-truss community including Ning Liu, Ying Li, and Jun Yan. Within this community, Weiguo Fan is not more impactful than other members and is not even directly connected to Ying*

[1]https://www.convosight.com/brands/community-marketing-case-study/amazon-check-kar-lena/index.html

Li. In contrast, Weiguo Fan is central and maintains direct connections with all members of the characteristic community (in blue). The characteristic community not only showcases his influence but is also more robust, encompassing his key frequent coauthors like Wei-Ying Ma and Benyu Zhang[2].

Motivated by the limitations of existing community search methods, we introduce the novel problem of *Characteristic cOmmunity Discovery* (COD) over attributed graphs. Given a query node q in an attributed graph g and a query attribute ℓ_q, COD finds the *characteristic community* of q, defined as the largest subgraph $C^*(q)$ of g that has high structure and attribute cohesiveness, as in [1]–[3], and in which q is one of the *top-k influential* nodes.

In order to analyze the influence of the query node across communities at different granularities, we consider the COD problem in the context of a *community hierarchy*. Such a hierarchical setting allows us to assess the node's influence comprehensively, from smaller, more cohesive subgraphs to larger, more intricate ones. Additionally, the hierarchical nature of communities provides an inherent ranking mechanism w.r.t. community sizes. Despite the rich literature on hierarchical graph clustering [15]–[20], as well as attributed [1]–[5] or influential community search [6]–[8], there are still two unique algorithmic challenges to discovering personalized characteristic communities on large-scale attributed graphs.

First, it requires evaluating *all* relevant communities in a hierarchy since the query node's impact in its hierarchical communities is *non-monotonic*: it may not be influential in a smaller community but become influential in larger communities. Unfortunately, even equipped with state-of-the-art *reverse reachable* (RR) set-based methods for influence estimation [21]–[24], the brute-force approach is still prohibitively expensive since there could be hundreds of hierarchical communities relevant to a query node. Based on the observation that the communities at deeper levels of the hierarchy are contained in those at shallower levels, we propose a *compressed COD evaluation* approach that does not need to find the top-k influential nodes in each community from scratch; instead, it progressively discovers the top-k nodes from deeper to shallower levels by reusing the RR sets for influence estimation. We show that the time complexity of this compressed approach is $O(\Theta \cdot \omega + |\mathcal{H}(q)|)$, which decouples the sampling cost on the original graph (i.e., $\Theta \cdot \omega$) from the number of hierarchical communities containing the query node (i.e., $|\mathcal{H}(q)|$).

Second, to the best of our knowledge, no hierarchical graph clustering method can support query attributes that reflect topics of user interest. One could craft a *global reclustering approach* by transforming the original graph into a weighted graph using any existing method for attributed community analysis [25], [26] to capture the relevance to the query attribute ℓ_q and obtain an ℓ_q-aware community hierarchy by feeding the resultant weighted graph to any hierarchical clustering algorithm. Unfortunately, the global approach tends to generate skewed hierarchical communities and thus cannot effectively discover characteristic communities for query nodes since even the smallest communities in the hierarchy will be excessively large for the query node to hold a significant impact. To avoid such drawbacks, we devise a *LOcal hierarchical REclustering* (LORE) method. It initializes a community hierarchy $\mathcal{T}$ on the original graph by ignoring attribute information. Then, using $\mathcal{T}$ as a reference, it computes a *reclustering score* to identify an appropriate local community $C \in \mathcal{T}$ that should be reclustered to provide a fine-grained attributed hierarchical community containing the query node. LORE also enables us to accelerate the processing of COD with a HIMOR index, which precomputes and maintains the influence ranks of nodes in $\mathcal{T}$ to avoid rank re-computations for communities that are not reclustered.

Our main contributions are summarized as follows.

- We introduce a novel problem of *characteristic community discovery* (COD) that finds the largest attributed community in which a query node is one of the top-k influential nodes (Section II).
- We propose the compressed COD evaluation approach that eliminates redundant influence evaluations for overlapped communities in a hierarchy and scales the computation to large graphs (Section III).
- We design LORE to support efficient COD processing over attributed graphs by finding fine-grained communities with local reclustering; we further develop an index-based method on top of LORE (Section IV).
- Through extensive experiments on six networks with real-world and synthetic node attributes, we show that our proposed methods discover larger and better characteristic communities than traditional community search methods; moreover, our fully optimized algorithm achieves up to a 25x speedup over well-crafted baselines (Section V).

II. Preliminaries

A. Notations and Problem Statement

Graphs. Let us consider a connected graph $g = (\mathcal{V}_g, \mathcal{E}_g)$ with node set $\mathcal{V}_g$ and edge set $\mathcal{E}_g$. We denote the set of neighbors of a node $v \in \mathcal{V}_g$ as $\mathcal{N}_g(v)$. For ease of presentation, we will drop the subscript g when it is clear from the context. We assume g to be an attributed graph and denote the set of attributes as $\mathcal{A}$. For node $v \in \mathcal{V}$, we use $\mathcal{A}(v)$ to denote the set of attributes of node v. In this paper, we focus on categorical attributes on nodes following the prior literature on attributed community search [1]–[5]. Nevertheless, our method can also be adapted to handle other types of attributes. In such cases, we employ embedding techniques to represent different attributes in our method. For example, various embedding techniques have been proposed for both text attributes [27], [28] and numerical attributes [29].

Hierarchical Communities. First, we emphasize that our framework accommodates any hierarchical graph clustering method [15]–[17] to obtain high-quality communities of increasing sizes, as it is orthogonal to the community hierarchy. Given a graph g, a *community* C is an *induced* subgraph of g,

[2]https://research.com/u/weiguo-fan,-patrick-

and $|C|$ denotes the number of nodes in C. The *community hierarchy* of g is a collection of communities $\mathcal{T} = \{C_0, C_1, \ldots, C_{|\mathcal{T}|-1}\}$ organized in a tree structure obtained by any hierarchical graph clustering method. The root vertex of $\mathcal{T}$ holds a community that comprises all nodes in g, and each leaf vertex holds a distinct node in g. A community $C_i \in \mathcal{T}$ is a parent/ancestor of $C_j \in \mathcal{T}$ iff C_j is a subgraph of C_i. For a community $C \in \mathcal{T}$, $\mathbf{dep}(C) \in \mathbb{Z}^+$ denotes the *depth* of C from the root vertex in $\mathcal{T}$. For a node $u \in \mathcal{V}$, we obtain a set of hierarchical communities $\mathcal{H}(u) = \{C_0(u), \ldots, C_{|\mathcal{H}(u)|-1}(u)\}$ by extracting all the communities in $\mathcal{T}$ that contain u. The communities in $\mathcal{H}(u)$ are sorted in *descending* order of depth, i.e., $\mathbf{dep}(C_i(u)) > \mathbf{dep}(C_j(u))$ when $i < j$ and $C_{|\mathcal{H}(u)|-1}(u) = g$. We use $\mathcal{H}(u|C)$ to denote the subset of $\mathcal{H}(u)$ containing the descendant communities of C. For a pair of nodes $u, v \in \mathcal{V}$, let $\mathbf{lca}(u, v)$ denote the smallest community in $\mathcal{T}$ that contains both u and v, i.e., their *lowest common ancestor* in $\mathcal{T}$. We also use $\mathbf{lca}(v)$ to denote the parent of v and further abuse $\mathbf{dep}(u, v)$ to denote $\mathbf{dep}(\mathbf{lca}(u, v))$ for ease of presentation.

Influence Models. Various models have been proposed to measure the influences of nodes in networks [30]–[32]. For ease of presentation, we take the independent cascade (IC) model [33] as an example. Nevertheless, our proposed method can support other typical influence models, such as the linear threshold (LT) model [34] and triggering models [35], with simple adaptions as long as they are compatible with RR set-based influence evaluation. Please refer to a survey [36] for how RR sets can be used for different influence models.

Under the IC model, each edge (u, v) is associated with an *influence probability* $p(u, v)$ denoting the influence exercised from u to v on g. Given a seed node $q \in \mathcal{V}$, a stochastic influence process unfolds in discrete steps. At step 0, q is activated. At each step $t > 0$, a node u activated in step $t-1$ tries to activate each inactive neighbor thereof, $v \in \mathcal{V}$, with probability $p(u, v)$. Each activated node has only one chance to activate each of its neighbors. The process ends when no more nodes can be activated. The *influence* $\sigma_g(q)$ of q in g is the expected number of activated nodes in the stochastic process. We use $\mathbf{rank}_g(q)$ to denote the *influence rank* of q in g, i.e., the number of nodes with a larger influence than q.

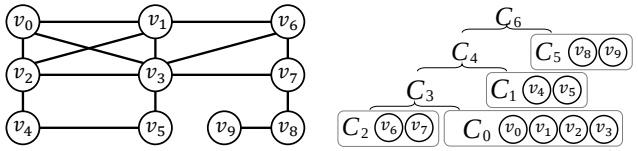
RR Set. As computing the influence of a node is #P-hard [37], an efficient method for influence estimation is to use the *reverse reachable* (RR) sets [21]. An RR set r_g in a graph g is generated as follows. At step 0, we sample a *source* node s uniformly at random from $\mathcal{V}$ and add it to r_g as an activated node. At each step $t > 0$, a node v activated at step $t-1$ has one chance to *reversely* activate each inactive neighbor thereof, $u \in \mathcal{V}$, with probability $p(u, v)$. The process ends when no new nodes can be activated. The following theorem indicates the correctness of the influence estimation using RR sets.

Theorem 1 (cf. [21])**.** *Given a node $q \in \mathcal{V}$, $\sigma_g(q) = p_g(q) \cdot |\mathcal{V}|$ where $p_g(q)$ is the probability that q appears in an RR set.*

Problem Statement. The problem of *Characteristic cOmmunity Discovery* (COD) finds the *largest* community in which a query node is one of the top-k influential nodes. We frame the problem with a *community hierarchy* since the hierarchy facilitates the examination of the query node's influence in communities of different sizes. The hierarchy can be obtained through any hierarchical clustering algorithm [15]–[20]. For ease of exposition, we introduce the COD problem on top of any given community hierarchy and defer the discussion on supporting query attributes to Section IV.

Definition 1 (COD)**.** *Given a graph g, a query node q, a set of hierarchical communities $\mathcal{H}(q)$ and the required influence rank k, find the characteristic community*

$$C^*(q) = \underset{C \in \mathcal{H}(q) : \mathbf{rank}_C(q) \leq k}{\arg\max} |C|. \tag{1}$$



(a) Exemplary graph g (b) Community hierarchy $\mathcal{T}$

Fig. 2. Exemplary graph g with corresponding community hierarchy $\mathcal{T}$.

Example 2. *Fig. 2 presents a graph g with 10 nodes and 15 edges. We use the IC model, where the influence probability of each edge is assigned by the weighted cascade model [38], for influence estimation. The community hierarchy on g is represented as $\mathcal{T} = \{C_0, C_1, \ldots, C_6\}$. Given two nodes v_0 and v_6, their lowest common ancestor is $\mathbf{lca}(v_0, v_6) = C_3$ with $\mathbf{dep}(C_3) = 3$. Given a query node v_0, the community hierarchy containing v_0 is $\mathcal{H}(v_0) = \{C_0, C_3, C_4, C_6\}$. The influence ranks of v_0 in those communities are $\mathbf{rank}_{C_0}(v_0) = 1$, $\mathbf{rank}_{C_3}(v_0) = 3$, and $\mathbf{rank}_{C_4}(v_0) = \mathbf{rank}_{C_6}(v_0) = 4$. Therefore, C_0 is the characteristic community $C^*(v_0)$ for $k = 1$.*

B. Related Work

Attributed Community Search (ACS). The ACS problem finds a densely connected subgraph containing one or more query nodes while ensuring that most nodes in the subgraph have the query attributes [1]–[5], [39], [40] (see [41] for a more comprehensive survey). For example, [2] and [3] return the maximal k-core and triangle-connected k-truss in which each node shares the query attributes, respectively. However, those studies focus on the structure and attribute coherence of the discovered community but do not consider the influence of the query node therein.

Tangentially to our work, another line of studies, known as *influential community search* (ICS) [6]–[8], aims to discover a community that has the maximum influence as a *whole* over the graph. This objective differs from the one of COD. Further, [42] studies an inverse problem of COD that discovers communities with strong influences on a query node.

Influence Maximization (IM). The problems surrounding network influences have been widely studied in the literature. Most existing works focus on the IM problem, which calls for a set of k seed nodes yielding the maximum influence. A plethora of studies has striven to enhance the efficiency of seed

selection (see [36] for a survey). However, those methods consider the influences of nodes on the entire network. Our work, by contrast, focuses on node influences within communities at different scales to discover the characteristic community of a query node. Even if the node is not globally influential, COD still finds a (local) community in which the node has a significant impact.

Hierarchical Graph Clustering (HGC). The HGC problem aims to recursively partition all the nodes in a graph into a community hierarchy. There are two types of methods for HGC. The first type adopts a divisive (top-down) approach [15], [43]–[45]. For instance, they partition the graph into hierarchically organized communities by iteratively removing edges with maximum betweenness [15]. The second type adopts an agglomerative (bottom-up) approach, which initializes every node as a community and iteratively merges two communities that maximize a linkage function [16], [19], [46], [47] until only one community remains. The HGC problem is orthogonal to the COD problem since we may employ any HGC method to obtain a community hierarchy. In this work, we employ the agglomerative approach due to its better theoretical and empirical efficiency over the divisive counterpart [17].

Summary. COD builds a strong connection between the above three areas of related work. For a given community hierarchy and a query node q, a solution to COD should consider both the influence of q and community cohesiveness to find the characteristic community of q.

III. Compressed COD Evaluation

Given a query node q and any hierarchical community $\mathcal{H}(q)$, the node q can be a top-k influential node in a larger community but not in a smaller sub-community. For example, Prof. Jiawei Han is arguably the most (top-1) influential researcher in the data mining community. But within a sub-community of data mining, such as graph neural networks (GNNs), another prominent researcher like Prof. Jure Leskovec, who specializes in this domain, may have a greater influence.

Lemma 1. *The influence rank* $\textbf{\textit{rank}}_C(q)$ *of the query node* q *is non-monotone with respect to* $\textbf{\textit{dep}}(C)$.

Based on Lemma 1, to find the correct characteristic community $C^*(q)$ exactly, we have to calculate the influence rank of q over all communities in $\mathcal{H}(q)$. Consequently, we devise a generic two-stage framework for COD processing. The first stage estimates the influence score $\sigma_C(v)$ of each node v in each $C \in \mathcal{H}(q)$. Then, the second stage finds the top-k influential nodes per community and returns the largest community where q is one of the top-k nodes.

A naïve approach following the generic framework would process each community independently. For each $C \in \mathcal{H}(q)$, we may generate an adequate number of RR sets to estimate the influence of each node in C and then scan all influence values to get the top-k nodes. As our experiments show, this naïve approach is prohibitively expensive on large graphs since there are too many communities in $\mathcal{H}(q)$.

We thus propose a *compressed COD evaluation* that eschews redundant influence and rank computations incurred by overlapping communities. We introduce an improved two-stage processing approach with *shared sample generation* and *incremental top-k evaluation*. Our theoretical analysis reveals that redundant computations are reduced to the minimum.

A. Shared Sample Generation

Motivated by the observation that communities in $\mathcal{H}(q)$ are nested with each other, we augment the RR set construction with a graph structure, the *RR graph*, which is shared across communities for efficient influence estimation.

Definition 2 (RR graph)**.** *Given an RR set r_g, its corresponding RR graph is a graph $R_g = (r_g, \mathcal{E}_{R_g})$ where $\mathcal{E}_{R_g}$ is the set of edges in $\mathcal{E}_g$ activated to generate r_g. We say that the source node of r_g is also the source node of R_g.*

Definition 3 (Induced RR graph)**.** *Given an RR graph R_g and a community C, we define the induced RR graph of R_g on C as $R_g(C) = (r_g \cap \mathcal{V}_C, \mathcal{E}_g \cap \mathcal{E}_C)$.*

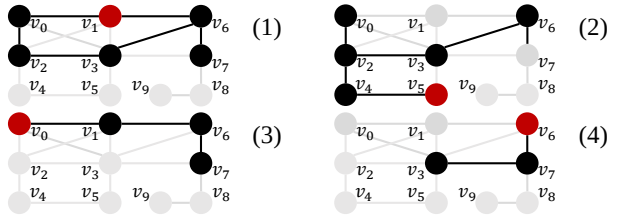
As the following theorem shows, an induced RR graph can be used to estimate influence correctly.

Theorem 2. *Given a node $q \in C$, $\sigma_C(q) = p_C(q) \cdot |C|$, where $p_C(q)$ is the probability that q is reachable from the source node in a randomly induced RR graph $R_g(C)$.*

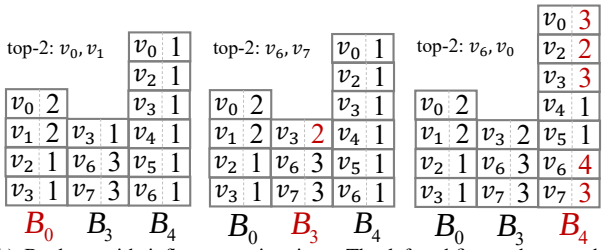
Proof. Let us consider a possible world g_i on graph g where each edge (u, v) is sampled to be independently activated with $p(u, v)$. For any source node $s \in C$, an RR set from s on C built w.r.t. g_i is identical to the set of nodes reachable from s by traversing nodes in C, i.e., an induced RR graph $R_g(C)$ where R_g is generated from s w.r.t. g_i. We conclude the proof by taking the expectation over all possible g_i. □

Based on Theorem 2, the influence of a node in different communities can be estimated by generating an adequate number of random RR graphs on g. Consequently, we propose the *hierarchical-first search* (HFS) algorithm that efficiently compresses the evaluation of influence values with induced RR graphs. For an instance of RR graph R_g, HFS starts the traversal from R_g's source node s. It first explores the *smallest* community $C \in \mathcal{H}(q)$ that contains s and uses the corresponding induced RR graph $R_g(C)$ to update the influence values of nodes. Subsequently, it iteratively explores increasingly larger communities and records node influences. Note that the influence value of a node v in R_g is recorded only in the community where HFS *first* visits v. Although its influence values in larger communities should also be recorded, those operations are delayed to allow efficient *incremental top-k evaluation* in Section III-B.

Example 3. *Fig. 3(a) shows four random RR graphs. We describe the HFS algorithm on the first two random RR graphs in detail. For RR graph (1), the traversal starts from the source node v_1. It first traverses the induced RR graph $R_g(C_0)$, where C_0 is as in Fig. 2. It explores nodes v_0, v_2 and $v_3 \in R_g(C_0)$*



(a) Examples of four random RR graphs. We mark the nodes and edges of each random RR graph in black and the source node in red.



(b) Buckets with influence estimations. The left subfigure denotes the buckets the HFS algorithm produces. The middle and right subfigures present the buckets during and after the incremental top-k evaluation process. Note that B_1 and B_2 are not used since C_1 and C_2 do not contain the query node v_0.

Fig. 3. Examples of random RR graphs and the corresponding buckets created for influence estimations.

and records their influences in bucket B_0. Then, HFS further traverses the induced RR graph $R_g(C_3)$, where C_3 is as in Fig. 2, explores nodes v_6, $v_7 \in R_g(C_3)$ and records their initial node influences in bucket B_3. For RR graph (2), the traversal starts from v_5. It traverses the induced RR graph $R_g(C_4)$. We have $R_g(C_0) = \varnothing$ and $R_g(C_3) = \varnothing$ in RR graph (2) since v_5 is not contained in C_0 and C_3, as Fig. 2 shows. We explore nodes v_4, v_2, v_0, v_3 and $v_6 \in R_g(C_4)$ and record their influences in bucket B_4. After HFS has processed all four RR graphs, we obtain the buckets shown in the first three columns of Fig. 3(b). For example, $B_0(v_1) = 2$ because v_1 appears in the induced RR graphs over C_0 of RR graphs (1) and (3). Also note that v_3 appears in three induced RR graphs (1), (2), and (3) over C_4. To recover the influence estimate $\sigma_{C_4}(v_3)$ for v_3 over C_4 from the buckets, we have $B_0(v_3) + B_3(v_3) + B_4(v_3) = 3$.

B. Incremental Top-k Evaluation

Given the influence values obtained by HFS, we need to check whether q is a top-k influential node in the communities of $\mathcal{H}(q)$. We propose an incremental evaluation algorithm that obtains the top-k nodes of each community in a *single pass* over the buckets. The algorithm scans the corresponding bucket to compute the top-k influential nodes in each community of $\mathcal{H}(q)$. It first scans bucket B_0 corresponding to the smallest community in $\mathcal{H}(q)$ and records the top-k nodes with the largest values therein. It then moves to the next bucket corresponding to a larger community. Still, instead of computing from scratch, it reuses the top-k results from the previous bucket and scans the items in the current bucket to update them. Theorem 3 guarantees the correctness of such an incremental approach by indicating that any node that does not appear in the current bucket and is not a previous top-k node *cannot* be a top-k node in the current bucket.

Theorem 3. *For a parameter k and a node v, if $\textbf{rank}_{C_h}(v) > k$ and $B_{h+1}(v)$ does not exist, then $\textbf{rank}_{C_{h+1}}(v) > k$.*

Proof. When samples are fixed, $\textbf{rank}_{C_h(q)}(v) > k$ is equivalent to the existence of k nodes u_0 to $u_{k-1} \in C_h(q)$ such that each of their estimated influences $\sum_{0\le j\le h} B_j(u_i) > \sum_{0\le j\le h} B_j(v)$. In community $C_{h+1}(q)$, since $B_{h+1}(v)$ does not exist, we have $\sum_{0\le j\le h+1} B_j(v) = \sum_{0\le j\le h} B_j(v) < \sum_{0\le j\le h} B_j(u_i) \le \sum_{0\le j\le h+1} B_j(u_i)$ for $0 \le i \le k-1$. Thus, $\textbf{rank}_{C_{h+1}(q)}(v) > k$. □

Example 4. *In Fig. 3(b), for a query node v_0 and $k = 2$, the incremental algorithm first scans bucket B_0 and records the top-2 nodes $\{(v_0, 2), (v_1, 2)\}$. Next, it scans bucket B_3. $B_3(v_6) = 3$ and $B_3(v_7) = 3$ are larger than the second value recorded in the previous top-2 nodes. Thus, we update the top-2 nodes after scanning B_3 to $\{(v_6, 3), (v_7, 3)\}$. Finally, it scans bucket B_4 and update values therein, e.g., $B_4(v_0)$ to $B_4(v_0) + B_0(v_0) = 3$. The recorded top-2 nodes are updated to $\{(v_6, 3), (v_0, 3)\}$. Note that, although v_1 exists in B_0, we do not need to consider v_1 when scanning B_4 because v_1 is contained neither in B_4 nor in the top-2 nodes in B_3. Hence, due to Theorem 3, it cannot be a top-2 node in B_4.*

C. Implementation and Complexity

Algorithm 1 depicts the pseudo-code of the overall *compressed COD evaluation* approach. It receives a graph g, the hierarchical communities $\mathcal{H}(q)$, a query node q, and a parameter k as input, and outputs the characteristic community $C^*(q)$ of q. It first generates Θ RR graphs, stores them in a collection $\mathcal{R}$, and adds the source node of each RR graph in a priority queue Q (Lines 1–6). Note that Q only has at most $|\mathcal{H}(q)|$ distinct priority values. Thus, $|\mathcal{H}(q)|$ hash maps are kept so that each insertion takes $O(1)$ time. It then performs HFS over all RR graphs and updates the buckets for each community in $\mathcal{H}(q)$ (Lines 7–15). Next, it performs an incremental top-k evaluation on each bucket (Lines 16–27). It uses a heap H to record the top-k nodes for each bucket. It will update the items in bucket B_h and add $v \in B_h$ to H if the updated $B_h(v)$ is larger than the activation count of the current top of H (Lines 19–25). If q is in H after scanning B_h, $C_h(q)$ is updated as its characteristic community (Lines 26&27). Finally, it returns $C^*(q)$ after processing all the communities in $\mathcal{H}(q)$.

We analyze the time complexity of Algorithm 1 as follows:

Lemma 2. *$\sum_{0\le h<|\mathcal{H}(q)|} |B_h|$ is bounded by $O(|\mathcal{R}|)$, where $|\mathcal{R}|$ denotes the number of nodes in $\mathcal{R}$.*

Proof. The shared sample generation of Algorithm 1 adds items to buckets while exploring a node in $\mathcal{R}$. HFS only explores a node in $\mathcal{R}$ once. Thus, the total size of the buckets is bounded by $O(|\mathcal{R}|)$ in the first stage. In the incremental top-k evaluation stage, the algorithm only accesses existing items in buckets and does not add items to buckets. Thus, the total size of the buckets during Algorithm 1 is $O(|\mathcal{R}|)$. □

Algorithm 1: Compressed COD Evaluation

Input: Graph g, hierarchical communities $\mathcal{H}(q)$, query node q, parameter k
Output: Characteristic community $C^*(q)$ of q

```
// Shared Sample Generation
Initialize an empty set R and an empty priority queue Q;
for i = 1 to Θ do
    Generate a random RR graph R from source v_s ∈ V;
    h ← |H(q)| − dep(v_s);
    Add (v_s, R) to Q[h];
    R ← R ∪ {R};
for h = 0 to |H(q)| − 1 do
    B_h ← ∅;
    while Q[h] ≠ ∅ do
        (v, R) ← Q[h].next();
        Update B_h(v);
        foreach u ∈ N_R(v) do
            if u is not explored then
                h' ← |H(q)| − min(dep(C_h), dep(u));
                Add (u, R) to Q[h'];
// Incremental Top-k Evaluation
Set τ(v) ← 0 for each v ∈ V;
Initialize H as an empty min-heap of size k;
for h = 0 to |H| − 1 do
    foreach v ∈ B_h do
        B_h(v) ← B_h(v) + τ(v);
        τ(v) ← B_h(v);
        if τ(H.top()) < B_h(v) and v ∉ H then
            Add v to H;
            if H.size() > k then
                H.pop();
    if q ∈ H then
        C*(q) ← C_h(q);
return C*(q);
```

Theorem 4. *The time complexity of the compressed COD evaluation is* $O(\Theta \cdot \omega + |\mathcal{H}(q)|)$.

Proof. HFS first generates a collection $\mathcal{R}$ of Θ RR graphs in $O(\Theta \cdot \omega)$ operations, where ω is the expected cost to generate an RR graph on g. The collection $\mathcal{R}$ is then traversed to update the influences in buckets. Note that each node in $\mathcal{R}$ is only explored once. Thus, this step requires $O(\text{vol}(\mathcal{R}) + |\mathcal{R}|)$ operations, where $\text{vol}(\mathcal{R})$ denotes the number of edges in $\mathcal{R}$. In the incremental top-k evaluation stage, we scan each bucket to obtain the top-k influential nodes therein. By Lemma 2, the number of operations is $O(|\mathcal{R}| + |\mathcal{H}(q)|)$, where $|\mathcal{H}(q)|$ operations are needed because the algorithm checks buckets for each $C \in \mathcal{H}(q)$ even if they are empty. Since $\Theta \cdot \omega \geq \text{vol}(\mathcal{R}) \geq |\mathcal{R}|$, the complexity is $O(\Theta \cdot \omega + |\mathcal{H}(q)|)$. □

Discussion. The preceding analysis shows that the compressed evaluation effectively eschews redundant computations among communities. While we need to check whether the query q is in the top-k for each community in $\mathcal{H}(q)$, the term $|\mathcal{H}(q)|$ is decoupled from the sampling cost in the complexity.

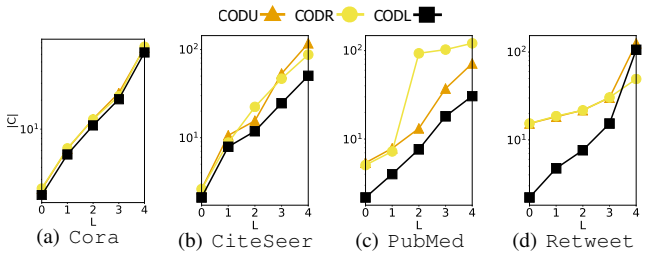


Fig. 4. The average size of the 5-deepest communities containing a given query node in the community hierarchy for CODU (non-attributed), CODR (global reclustering), and CODL (local reclustering) across four datasets.

IV. Attributed Community Hierarchy

In this section, we discuss how to support query attributes in COD. By indicating a query attribute ℓ_q, a user aims to find a characteristic community on a desired topic, i.e., with high density on attribute ℓ_q [1]–[3]. For instance, to organize an academic conference on a certain research area, one may send invitations to a characteristic community that comprises researchers in the area. To our best knowledge, no prior work considers query attributes in community hierarchy construction. Still, one can devise a *global reclustering* approach:

- Use any method for attributed community analysis [25], [26] to assign edge weights to reflect the synergized information of the topology and query attribute ℓ_q, generating a weighted graph g_ℓ;
- Partition g_ℓ into hierarchical communities and utilize the approach in Section III to get $C^*(q)$.

However, the above global reclustering approach typically fails to provide fine-grained hierarchical communities for query nodes with limited influences over g due to its tendency to generate skewed hierarchical communities. This occurs because hierarchical clustering algorithms are often dominated by hub nodes closely connected to a large fraction of nodes in g, and the hierarchical communities are thus formed by attaching nodes to these hubs. Consequently, a query node with limited influence gets assigned to large communities around hubs, and even the smallest community is still too large for a query node to be impactful. As a result, such query nodes cannot find their characteristic community through the global reclustering approach. To support the above claim, Fig. 4 presents the average size of the 5-deepest (i.e., 5-smallest) communities containing a given query node in the community hierarchies obtained by three different methods. Both community hierarchies generated by global clustering on non-attributed graphs and global reclustering on attribute-weighted graphs exhibit large community sizes, especially on the PubMed and Retweet datasets.

To address the drawback of global reclustering, we propose a *local* reclustering approach that only reclusters a local subgraph closely related to the query node q and thus alleviates the impact of global hubs.

A. Local Hierarchical Reclustering

To mitigate the deficiency of global reclustering, we propose a *LOcal hierarchical REclustering* (LORE) algorithm that par-

tially reclusters the graph to update the community hierarchy. In particular, LORE first identifies an appropriate subgraph C_ℓ containing the query node q in the non-attributed community hierarchy $\mathcal{H}(q)$ (by ignoring attributes) and then reclusters C_ℓ by the global reclustering approach. The reclustered communities substitute C_ℓ and its descendants in $\mathcal{H}(q)$ to obtain the *attribute-aware* community hierarchy $\mathcal{H}_\ell(q)$.

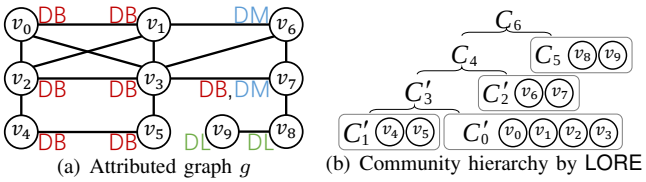
The core of LORE is choosing the community C_ℓ on which the reclustering should be performed. We propose a novel *reclustering score* $r(C)$ for each community $C \in \mathcal{H}(q)$ to measure the extent to which C needs to be reclustered according to ℓ_q and select the community with the maximum score as $C_\ell = \arg\max_{r(C)} C$. Intuitively, if a community C is closely related to the query attribute, it should be reclustered to reflect such a closeness. As such, we introduce a reclustering score based on attributed edge density, tailored explicitly for hierarchical communities:

Definition 4. *Given a query attribute ℓ_q, a non-attributed community hierarchy $\mathcal{H}(q)$, and a community $C \in \mathcal{H}(q)$, the* reclustering score *of C is defined as:*

$$r(C) = \frac{\sum_{(u,v)\in\delta(q,C)} \textbf{\textit{dep}}(u,v)}{|C|}, \tag{2}$$

where $\delta(q, C)$ is the set of query-attributed *edges (u, v) such that both u and v have the query attribute ℓ_q and (u, v) is divided by the descendants of C in $\mathcal{H}(q)$.*

Note that the numerator of $r(C)$ sums over query-attributed edges, each of which is separated into different clusters by the descendants of C in $\mathcal{H}(q)$. Since LORE aims at reclustering a community closely related to the query node, instead of simply counting such edges, we sum their depths, $\textbf{dep}(u, v)$. This summation measures the proximity of edges to the query node q. If an edge is divided at a deeper level of $\mathcal{H}(q|C)$, it is considered closer to the query node q. As illustrated in Fig. 4, LORE produces smaller and more fine-grained communities at deeper levels of the community hierarchy, which can better identify the characteristic community for any query node.



(a) Attributed graph g (b) Community hierarchy by LORE

Fig. 5. The graph g augmented with node attributes. Given a query attribute $\ell_q = \{\texttt{DB}\}$ and a query node v_0, C_4 in Fig. 2 is reclustered by LORE.

Example 5. *Fig. 5 illustrates the graph of Fig. 2 augmented with node attributes. Given a query node v_0, $\delta(v_0, C_4) = \{(v_2, v_4), (v_3, v_5), (v_3, v_7)\}$ where (v_3, v_7) is divided by C_0 and $(v_2, v_4), (v_3, v_5)$ are divided by C_3. The reclustering score of C_4 in Fig. 2 is thus $r(C_4) = \frac{\sum_{(u,v)\in\delta(v_0,C_4)} \textbf{\textit{dep}}(u,v)}{|C_4|} = \frac{2+2+3}{8} = \frac{7}{8}$.*

Efficient Score Computation. In addition to the expressiveness of $r(C)$, a recursion formulation indicates an opportunity for efficient computation as follows:

$$r(C) = \frac{r(C') \cdot |C'| + \Delta(C') \cdot \textbf{dep}(C')}{|C|}, \tag{3}$$

where C' is the child of C in $\mathcal{H}(q)$ and $\Delta(C')$ denotes the number of query-attributed edges divided by C'. Since $\Delta(C')$ can be efficiently computed by finding the lowest common ancestor of any query-attributed edges in C', all reclustering scores are obtained in a linear time w.r.t. the original graph g.

Algorithm 2 shows the pseudo-code of LORE. It receives the graph g, a query node q, and a query attribute ℓ_q, and the community hierarchy $\mathcal{T}$ generated by non-attributed hierarchical clustering over g as input and outputs a community hierarchy $\mathcal{H}_\ell(q)$ related to the query attribute. It first identifies the community C_ℓ to recluster by QueryAttrRelated (Line 1) and then reclusters C_ℓ through hierarchical clustering according to weights in g_ℓ (Lines 2–3). Finally, the algorithm returns the union of attribute-aware communities in C_ℓ and the ancestors of C_ℓ in the non-attributed community hierarchy $\mathcal{T}$ (Line 4) as the updated community hierarchy $\mathcal{H}_\ell(q)$ for q. To identify C_ℓ, QueryAttrRelated checks each query-attributed edge and, if the edge is cut by an ancestor of q, updates the corresponding value in Δ (Lines 7–11). It computes the *reclustering score* of each community $C \in \mathcal{H}(q)$ in Lines 12–17. The variable r maintains the highest reclustering score to return the community of highest score as C_ℓ.

Algorithm 2: LORE

Input: Graph g, query (q, ℓ_q), community hierarchy $\mathcal{T}$
Output: Attribute-aware hierarchical communities $\mathcal{H}_\ell(q)$

1 $C_\ell \leftarrow$ QueryAttrRelated$(q, \ell_q, \mathcal{T})$;
2 $S_\ell \leftarrow$ subgraph of g_ℓ induced by C_ℓ;
3 $\mathcal{T}_\ell \leftarrow$ HierarchicalClustering(S_ℓ);
4 $\mathcal{H}_\ell(q) \leftarrow \text{Ancestors}(q, \mathcal{T}_\ell) \cup \text{Ancestors}(C_\ell, \mathcal{T})$;
5 **return** $\mathcal{H}_\ell(q)$;
6 **Function** QueryAttrRelated$(q, \ell_q, \mathcal{T})$**:**
7 Initialize an array Δ of length $|\mathcal{H}(q)|$;
8 **foreach** query-attributed edge (u, v) **do**
9 $d \leftarrow \textbf{dep}(u, v)$;
10 **if** $q \in \textbf{\textit{lca}}(u, v)$ **then**
11 $\Delta[d] \leftarrow \Delta[d] + 1$;
12 $S \leftarrow 0, r \leftarrow 0$;
13 $C_\ell \leftarrow \varnothing$;
14 **for** $i = 1$ **to** $|\mathcal{H}(q)| - 1$ **do**
15 $S \leftarrow \Delta[i] \cdot (|\mathcal{H}(q)| - i) + S$;
16 **if** $r < \frac{S}{|C_i(q)|}$ **then**
17 $r \leftarrow \frac{S}{|C_i(q)|}$ and $C_\ell \leftarrow C_i(q)$;
18 **return** C_ℓ;

Example 6. *In Fig. 5, for a given query attribute $\ell_q = \{\texttt{DB}\}$ and query node v_0, we first compute the reclustering score of communities above C_0 containing v_0 in Fig. 2. We have $\Delta(C_3) = 1$ and $\Delta(C_4) = 2$. Hence, the reclustering score of C_3 is $r(C_3) = \frac{\Delta(C_3)\cdot \textbf{\textit{dep}}(C_3)}{|C_3|} = \frac{3}{6} = \frac{1}{2}$. We compute $r(C_4)$ from $r(C_3)$ as $r(C_4) = \frac{r(C_3)\cdot|C_3| + \Delta(C_4)\cdot \textbf{\textit{dep}}(C_4)}{|C_4|} = \frac{3+2\cdot 2}{8} = \frac{7}{8}$.*

Since $r(C_4) > r(C_3)$, we select C_4 as C_ℓ for reclustering according to the weights in g_ℓ. Fig. 5 shows the result of local reclustering for C_4: the new communities C'_0 to C'_3. Thus, we obtain the community hierarchy $\mathcal{H}_\ell(v_0) = \{C'_0, C'_3, C_4, C_6\}$ that reflects query attribute information.

Theorem 5. *The reclustering scores of all communities in $\mathcal{H}(q)$ are computed in $O(|\mathcal{E}_g|)$ time for a query attribute ℓ_q.*

Proof. According to the definition of $r(C)$, we can obtain its recursive formulation as:

$$r(C) \cdot |C| = r(C') \cdot |C'| + \Delta(C') \cdot \mathbf{dep}(C'), \qquad (4)$$

where C' is the child of C in $\mathcal{H}(q)$ and $\Delta(C)$ denotes the number of query-attributed edges divided by C. We initialize an array Δ of zeros for each ancestor community of q in the non-attributed community hierarchy. Then, we traverse $\mathcal{E}_g$, and for each query-attributed edge, we add $\Delta(c)$ by one, where c is the lowest common ancestor of the query-attributed edge. This step takes $O(|\mathcal{E}_g|)$ time since the lowest common ancestor can be obtained in $O(1)$ time [48]. After obtaining Δ, we compute $r(C)$ for all communities in the non-attributed communities according to Eq. 4 in $O(|H(q)|)$ time. Note that there are at most $\mathcal{N}_g$ different communities containing q. Thus, $|\mathcal{H}(q)| \leq \mathcal{N}_g \leq |\mathcal{E}_g|$, and the reclustering scores of all communities in $\mathcal{H}(q)$ are computed in $O(|\mathcal{E}_g|)$ time. □

B. HIMOR-Index

Given the attribute-aware community hierarchy $\mathcal{H}_\ell(q)$ obtained by LORE, one can directly apply the compressed COD evaluation in Section III on $\mathcal{H}_\ell(q)$ to generate the characteristic community $C^*(q)$ for the query node q. However, processing $\mathcal{H}_\ell(q)$ containing very large communities for different query attributes is still time-consuming.

Notably, the attribute-aware $\mathcal{H}_\ell(q)$ returned by LORE is only affected by the query attribute at deeper levels (the descendants of C_ℓ), whereas the larger communities therein (the ancestors of C_ℓ) are unaffected, as they arise from the non-attributed community hierarchy. We exploit this invariant with the HIMOR-index, which stores the influence ranks of each node in the non-attributed community hierarchy. For a query node q, we first check whether there exists an ancestor C' of C_ℓ in the non-attributed community hierarchy within which q is a top-k influential node and return C' as $C^*(q)$ if it exists. Otherwise, we compute the $C^*(q)$ by compressed COD evaluation over the community hierarchy within C_ℓ.

Algorithm 3 presents the pseudo-code of COD query processing based on the HIMOR-index. It receives a graph g, a query (q, ℓ_q), the parameter k, and the HIMOR-index $\mathcal{I}_q$ that contains the influence rank of q in the non-attributed community hierarchy, and outputs the attributed-aware characteristic community $C^*(q)$. It traverses items in $\mathcal{I}_q$ corresponding to communities containing C_ℓ in $\mathcal{H}(q)$ in a top-down manner (Lines 1–2). If there doesn't exist a community in $\mathcal{H}(q)$ above C_ℓ such that q is top-k influential, the compressed COD framework is applied over the community hierarchy in $\mathcal{H}_\ell(q)$ below C to obtain $C^*(q)$ (Lines 3–4).

Algorithm 3: HIMOR-based COD

Input: Graph g, query (q, ℓ_q), parameter k, and the HIMOR-index $\mathcal{I}_q$
Output: Characteristic community $C^*(q)$ of q

1 **foreach** $(\tau, C) \in \mathcal{I}_q$ *and* $C_\ell \subseteq C$ **do**
2 **if** $\tau \leq k$ **then return** C;
3 Run Algorithm 1 on $\mathcal{H}_\ell(q|C_\ell)$ to get $C^*(q)$;
4 **return** $C^*(q)$;

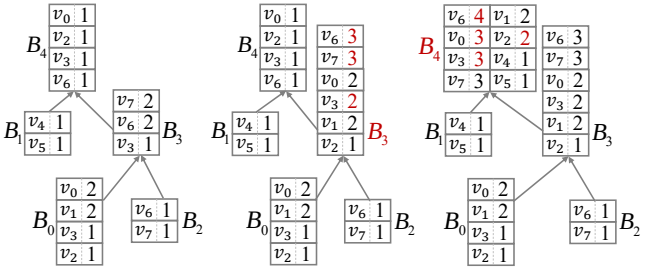


Fig. 6. Tree-structured buckets in the HIMOR index.

Compressed HIMOR Construction. We then propose a compressed construction algorithm for HIMOR. The key observation is that the community hierarchies for different nodes highly overlap at shallow levels. For two nodes $u, v \in \mathcal{V}$, the community hierarchies $\mathcal{H}(u)$ and $\mathcal{H}(v)$ will both contain ancestor communities of $\mathbf{lca}(u, v)$ in $\mathcal{T}$. Compressed HIMOR construction extends the compressed COD evaluation in Section III with two crucial differences:

- HIMOR estimates the influences of nodes over the tree-structured $\mathcal{T}$ rather than the linear-structured $\mathcal{H}_\ell(q)$;
- HIMOR computes all node ranks in each community rather than selecting the top-k nodes for a fixed k.

The compressed construction algorithm works as follows. *First*, it generates Θ RR graphs and applies the HFS algorithm for influence estimation on $\mathcal{T}$. For each RR graph generated from source node s, HFS uses a priority queue to explore the communities in $\mathcal{H}(s)$, from smaller to larger. To enable $O(1)$ insertion time for the queue, we maintain $|\mathcal{T}|$ hash maps as in Algorithm 1. When exploring node v tagged with C_i in the queue, HFS records the influence of v in bucket B_i, visits each $u \in \mathcal{N}(v)$, and adds $(u, \mathbf{lca}(u, C_i))$ to the queue, if u is unexplored, where $\mathbf{lca}(u, C_i)$ is the *smallest* community containing a path from s to u. The buckets are organized in a tree hierarchy corresponding to the community hierarchy in $\mathcal{T}$. *Second*, the algorithm processes buckets produced by HFS in a bottom-up manner. The processing of each bucket B_i proceeds in three steps: (1) update $B_i(v)$ to $B_i(v) + B_j(v)$ for each node v that appears in both B_i and a child thereof, B_j; (2) sort B_i in decreasing order; (3) merge B_i's child buckets with B_i to get the influence ranks of nodes in B_i. We construct the HIMOR-index by storing the influence rank of each node v in the selected communities.

Example 7. *The left subfigure of Fig. 6 shows the tree of buckets generated by HFS. Bucket B_4 contains items $(v_2, 1)$, $(v_3, 1)$, $(v_0, 1)$ and $(v_6, 1)$ recorded by applying HFS on RR graph (2) in Fig. 3(a). Note that items $(v_5, 1)$ and $(v_4, 1)$ are*

recorded in bucket B_1 and are incrementally added to B_4 in the second stage. Initially, we merge B_0 and B_2 into B_3. Since v_7 appears in both B_3 and B_2, we get $B_3(v_7) = 3$. Similarly, we obtain $B_3(v_6) = 3$ and $B_3(v_3) = 2$. The up-to-date bucket B_3 is then sorted as $B_3 = \{(v_7, 3), (v_6, 3), (v_3, 2)\}$. Then, we combine B_0 and B_2 with B_3 by performing a merge-sort *to obtain the influence ranks of the nodes in C_3. The middle subfigure in Fig. 6 shows the buckets after processing B_3. We process B_4 similarly, as shown in the right subfigure of Fig. 6.*

The following theorem gives the time complexity of the compressed HIMOR construction algorithm.

Theorem 6. *The time complexity of compressed HIMOR construction is $O(\Theta \cdot \omega + |\mathcal{R}| \cdot \log |\mathcal{V}| + \sum_{v \in \mathcal{V}} \textbf{\textit{dep}}(v))$.*

Proof. The first stage of compressed HIMOR construction requires $O(\Theta \cdot \omega + |\mathcal{R}|)$ operations, as for the influence estimation stage in compressed COD evaluation for query processing. The HFS algorithm requires an **lca** operation to compute the community tag for each node added into the queue. Following the indexing method in [48], each **lca** operation is executed in $O(1)$ time. In the second stage, to process B_i, the algorithm updates the values in B_i and sorts the updated B_i in $O(|B_i| \cdot \log |B_i|)$ operations. Then, the algorithm merges B_i with its child buckets. The node v is merged $\textbf{dep}(v)$ times. Thus, the time complexity of the second stage is $O(\sum_{B_i} |B_i| \cdot \log |B_i| + \sum_{v \in \mathcal{V}} \textbf{dep}(v)) = O(|\mathcal{R}| \cdot \log |\mathcal{V}| + \sum_{v \in \mathcal{V}} \textbf{dep}(v))$. The total time complexity of compressed HIMOR construction is $O(\Theta \cdot \omega + |\mathcal{R}| \cdot \log |\mathcal{V}| + \sum_{v \in \mathcal{V}} \textbf{dep}(v))$. □

Discussion. Real-world graphs are often highly dynamic in nature. Updates to graphs have an impact on the structure of hierarchical communities and the process of influence propagation. Several prior studies have studied the hierarchical clustering [49], [50] and influence estimation [51]–[53] in dynamic settings. Therefore, it holds promise to combine these approaches to implement the HIMOR-Index in the context of dynamic graphs. Nevertheless, incorporating them into the dynamic HIMOR-Index is not trivial since the compressed computation of the influences over the community hierarchy cannot be updated efficiently. We will continue to explore this issue in future work.

V. Experimental Study

A. Experimental Setup

Datasets. We conduct all the experiments on six real-world network datasets. The statistics of these networks are reported in Table I. Note that $|\bar{\mathcal{H}}_\ell(q)|$ refers to the average number of hierarchical communities that contain a query node.

Datasets `Cora`, `CiteSeer`, `PubMed`, and `Retweet`, are networks with real-world node attributes available at the Network Repository[3]. The remaining three datasets, `Amazon`, `DBLP`, and `LiveJournal`, are real-world networks without node attributes. We augment them with node attributes as in [1], [26]: we generate a set of $|\mathcal{A}| = 0.0001 \cdot |\mathcal{V}|$ distinct attributes and assign the same random attribute from $\mathcal{A}$ to all nodes in each ground-truth community defined by product categories in `Amazon`, publication venues in `DBLP` and user groups in `LiveJournal`. We employ the first six datasets for effectiveness and efficiency experiments and the large dataset `LiveJournal` for scalability tests.

TABLE I
NETWORK STATISTICS.

Network	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{A}\rvert$	$\lvert\bar{\mathcal{H}}_\ell(q)\rvert$
`Cora`	2,485	5,069	7	18.5
`CiteSeer`	2,110	3,668	6	18.9
`PubMed`	19,717	44,327	3	34.2
`Retweet`	18,470	48,053	2	165.3
`Amazon`	334,863	925,872	33	54.8
`DBLP`	317,080	1,049,866	31	47.9
`LiveJournal`	3,997,962	34,681,189	400	65.9

Query Generation. For each dataset, we randomly select 100 nodes as queries. For each query node q, we randomly pick one of its attributes as the query attribute.

Compared Methods. To our best knowledge, no prior work addresses the COD problem. Thus, we compare our approaches to existing methods for community search on attributed graphs, which aim to find a cohesive subgraph that contains the query node and has high attribute density, to the extent that the returned community satisfies our definition of a characteristic community.

- ACQ [2] finds a k-core containing the query node such that all nodes in the k-core share the query attribute.
- ATC [1] finds a (k, d)-truss containing the query node and maximizes a dedicated attribute score function.
- CAC [3] finds a triangle-connected k-truss containing the query node in which all nodes share the query attribute.

We compare different variants of the hierarchical approaches for the COD problem proposed in this work. Our implementation is publicly available on GitHub[4].

- CODU generates *non-attributed* hierarchical communities $\mathcal{H}(q)$ by agglomerative hierarchical clustering on g and then obtains $C^*(q)$ from $\mathcal{H}(q)$.
- CODR transforms g to a weighted graph g_ℓ by placing additional weights for query attributed edges and then performs hierarchical clustering over g_ℓ from scratch to generate the attributed-aware hierarchical communities $\mathcal{H}_\ell(q)$, from which $C^*(q)$ is obtained.
- CODL generates $\mathcal{H}_\ell(q)$ using LORE and obtains $C^*(q)$ from $\mathcal{H}_\ell(q)$ utilizing the HIMOR index.

Both CODU and CODR obtain $C^*(q)$ by compressed COD evaluation, as CODL does, since the naïve approach is time-prohibitive. We use the nearest neighbor chain algorithm [54], [55] and the unweighted-average linkage function [45] for agglomerative hierarchical clustering in all our approaches (CODU, CODR, and CODL). The choice of the linkage function as well as the transformation scheme to obtain g_ℓ is orthogonal to our contributions. Our methods can also be

[3]https://networkrepository.com/labeled.php

[4]https://github.com/YudongNiu/COD

combined with more sophisticated schemes [25], [26] and other linkage functions [16].

Parameter Setting. We set $k = 5$ as the default influence rank for COD queries and $\theta = 10$ as the default number of RR graphs generated from *each* node for influence estimation. We adopt the weighted cascade model [37], [56], setting the influence probability $p(u, v)$ for each $(u, v) \in \mathcal{E}$ to $\frac{1}{|\mathcal{N}(v)|}$.

Evaluation Measures. We quantify the effectiveness of each method with three measures on the characteristic community C^*. Among these, topology density $\rho(C^*)$ and attribute density $\varphi(C^*)$ are used in previous works [26], [57]–[59].

- **Size** $|C^*|$: the number of nodes in $C^*(q)$;
- **Topology density** $\rho(C^*)$: the ratio between the number of edges and the number of *node pairs* in $C^*(q)$;
- **Attribute density** $\varphi(C^*)$: the number of query attributes in $C^*(q)$ divided by the number of nodes in $C^*(q)$.

We report the average of each measure over 100 queries for each method. In case a community search method does not return a characteristic community of a query, i.e., the query node is not top-k influential in the community, we assign 0 to each measure.

B. Effectiveness Evaluation

We evaluate the effectiveness of our proposed method CODL against attributed community search methods (CAC, ACQ, and ATC) and the variants of our hierarchical approaches (CODU and CODR). The purpose is to demonstrate: (1) the effectiveness of hierarchical approaches for the COD problem (vs. CAC, ACQ, and ATC); (2) the effectiveness of the LORE algorithm (vs. CODU and CODR).

Average Size $|C^*|$**.** Figs. 7(a)–7(f) report the average size of $C^*(q)$ obtained by different methods on each dataset. The result shows that our hierarchical methods (CODU, CODR, and CODL) find significantly larger $C^*(q)$ for users than CAC, ACQ, and ATC. The reason is that the community search methods do not consider the influence of query node q when searching for the resulting community and thus cannot find the characteristic community for many queries. We also observe that the average size of $C^*(q)$ obtained by different methods increases monotonically with k. With a looser requirement on the influence rank of the query node q, larger characteristic communities are naturally discovered.

Average Topology Density $\rho(C^*)$**.** Figs. 7(g)–7(l) show the average topology density $\rho(C^*)$ for different methods across six datasets. The results indicate that our proposed method CODL achieves the largest $\rho(C^*)$ in most circumstances. *First*, CODL outperforms CODU and CODR across all the datasets. CODL achieves the peak topology density value when $k = 2$ and outperforms the baselines by 71.62% on average. CODR and CODU do not achieve comparable $\rho(C^*)$ to CODL since they do not return characteristic communities for many query nodes. The main reason is that their generated hierarchical communities are often highly skewed, i.e., even the deepest community containing q is still relatively large, as shown in Fig. 4, in which the query node cannot be a top-k influential node. The value of $\rho(C^*)$ decreases when $k > 2$ since larger and sparser characteristic communities will be discovered with a looser requirement on the influence rank of the query node q. *Second*, CODL achieves larger $\rho(C^*)$ than CAC, ACQ, and ATC over four out of six datasets. For a small k, many communities found by these community search methods are not characteristic communities w.r.t. k, and thus lower density values are exhibited. When k is bigger, ATC and CAC achieve larger $\rho(C^*)$ than CODL on `CiteSeer` and `DBLP`. The reason is that ATC and CAC adopt the strict community models of (k, d)-truss and triangle-connected truss and thus return small and dense communities. With a looser restriction on the influence rank k, both baselines can find eligible characteristic communities with high values of $\rho(C^*)$. Nevertheless, the communities $C^*(q)$ obtained by ATC and CAC on `CiteSeer` and `DBLP` are much smaller than those obtained by CODL according to Fig. 7(b) and Fig. 7(f).

Average Attribute Density $\varphi(C^*)$**.** Figs. 7(m)–7(r) depict the average attribute density $\varphi(C^*)$ of different methods on each dataset. The results again confirm the superiority of CODL: it achieves better or comparable values of $\varphi(C^*)$ against all baselines in most cases. We also have an additional observation that CODR achieves better performance than CODU consistently since it places more emphasis on the query attributes by reclustering the transformed graph g_ℓ.

Query Node Influence $I(q)$**.** Figs. 7(s)–7(x) present the average influence $I(q)$ of the query nodes on g for which the method returns a characteristic community for the query node. *First*, the average query node influence $I(q)$ generally decreases with increasing k. The reason is that, for a smaller k, characteristic communities can only be discovered for query nodes with relatively large influences on g due to the strict requirement on the influence rank of the query node. *Second*, CODL discovers characteristic communities for query nodes with the smallest $I(q)$ on average in most circumstances. Especially, CODL finds characteristic communities for query nodes with influences much smaller than CODR and CODU on `PubMed` and `Retweet`. This is because the hierarchical communities generated for CODL are locally reclustered around q, and thus CODL can provide more fine-grained communities containing q, within which q can be influential even if it is not influential over the entire g. Moreover, hierarchical-based methods usually discover characteristic communities for query nodes with smaller influences than community search methods (over 4 out of 6 datasets) since community search methods do not consider the influence of the query node at all when searching for the community and thus the resultant community can only be a characteristic community if the query node is globally influential. Among community search methods, ACQ usually requires the query node to be highly influential to discover a characteristic community since ACQ uses a core-based community model and tends to find very large communities within which only globally influential nodes can hold strong influences.

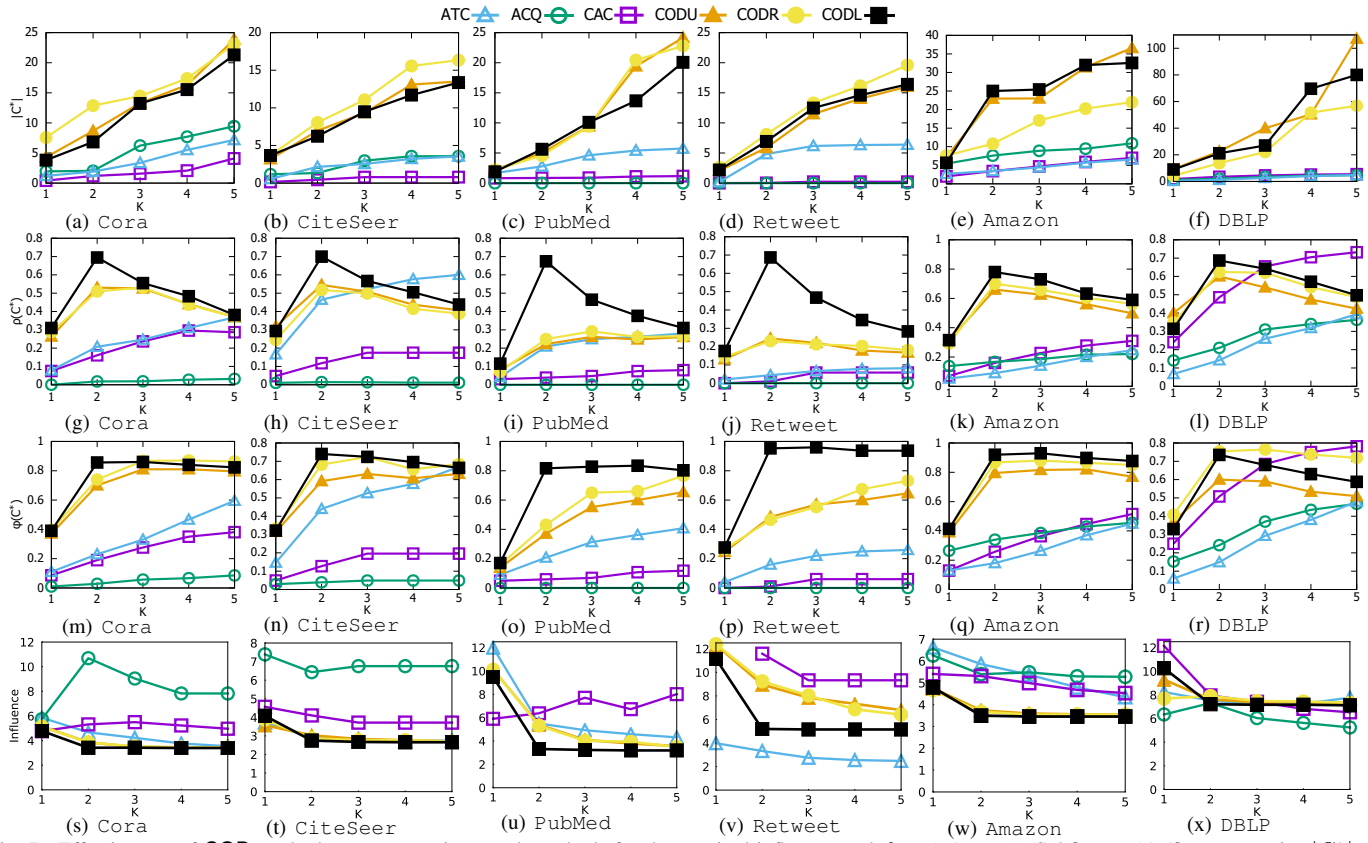


Fig. 7. Effectiveness of COD methods vs. community search methods for the required influence rank $k = 1, 2, \ldots, 5$. Subfigures (a)-(f): average size $|C^*|$. Subfigures (i)-(l): average topology density $\rho(C^*)$. Subfigures (m)-(r): average attribute density $\varphi(C^*)$. Subfigures (s)-(x): influence $I(q)$ of query nodes on g by taking the average over query nodes for which each method can discover a characteristic community. ACQ is not presented in (u) and (v) since it cannot find the characteristic community for any query node in PubMed and Retweet.

C. Compressed COD Evaluation

We evaluate the effectiveness of compressed COD evaluation in Section III by juxtaposing the following two methods:

- Compressed, a CODR variant using compressed evaluation to discover $C^*(q)$, generating $\Theta = \theta \cdot |\mathcal{V}|$ samples for influence estimation on all communities in $\mathcal{H}_\ell(q)$.
- Independent, another CODR variant that independently evaluates the influence rank of the query node in each community within $\mathcal{H}_\ell(q)$, using $\Theta = \theta \cdot \sum_{C \in \mathcal{H}_\ell(q)} |C|$ samples for influence estimation.

We aim to show that compressed COD evaluation significantly decreases sampled RR sets while retaining an accurate influence estimation. We study the average size and top-k precision of characteristic communities discovered by Compressed and Independent, where the top-k precision denotes the percentage of discovered communities in which the query node q is truly top-k influential, determined by sampling 1,000 RR sets for each node in the discovered community. We focus on Cora and CiteSeer, as Independent is prohibitively expensive on larger graphs.

Figs. 8(a) & 8(d) and Figs. 8(b) & 8(e) show the average top-k precision and size of C^* discovered by Compressed and Independent, respectively, for different numbers of samples in influence estimation. We observe that Compressed achieves better top-k precision but relatively

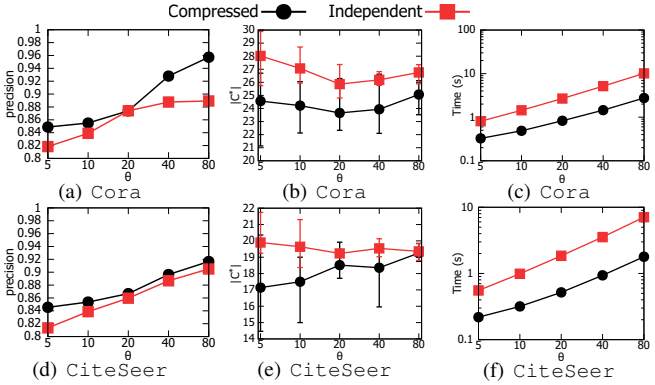


Fig. 8. Compressed vs. Independent for different θ's on Cora and CiteSeer. Subfigures (a) and (d): average top-k precision. Subfigures (b) and (e): average sizes of returned $C^*(q)$ with error bars indicating maximum and minimum sizes. Subfigures (c) and (f): execution time.

smaller sizes of C^* than Independent on both datasets. First, since Independent returns larger characteristic communities than Compressed, it is harder for Independent to estimate the influence rank of q precisely, which requires influence estimations of all nodes in the communities. Second, the reason why Independent tends to find larger characteristic communities than Compressed is due to sample correlations. Since Compressed reuses random RR sets across communities for influence estimation, once q is erroneously excluded from the top-k in a community, it will also be excluded from

the top-k in adjacent descendants of the community with high probability, and thus a smaller characteristic community is returned. In comparison, Independent samples RR sets for each community independently, and thus the false exclusion in one community will not influence its descendants. Note that the chance for an algorithm to decide if q is a top-k influential node erroneously is positively correlated with the influence difference between q and the k-th influential node in the community and, for a randomly sampled query node q, such a difference is often large if q is not a top-k node, whereas if q is indeed a top-k node, the influence gap is typically much smaller, which might lead to false exclusion.

Figs. 8(c) and 8(f) present the execution time of Compressed and Independent by varying the number of samples. We observe that (1) Compressed is remarkably more efficient than Independent; (2) the speedup ratio of Compressed over Independent remains nearly constant w.r.t. θ. Specifically, when $\theta = 80$, Compressed takes 2.76 and 1.79 seconds for COD processing on Cora and CiteSeer, respectively, whereas Independent uses 10.14 and 7.08 seconds accordingly. On larger datasets, Compressed achieves higher speedups. For example, Compressed runs 10 and 9 times faster than Independent on PubMed and Retweet, respectively. Furthermore, Independent does not finish within 36 hours on Amazon and DBLP when $\theta = 10$. Since Independent has to evaluate the influence rank of the query node in each community independently, its time consumption is severely affected by $|\mathcal{H}_\ell(q)|$. In contrast, $|\mathcal{H}_\ell(q)|$ is decoupled from the sampling cost in Compressed, and it only takes 132.03 and 196.34 seconds for Compressed to process a COD query on average on Amazon and DBLP, respectively. In Section V-D, we will show that the execution time is further reduced with optimizations.

D. Efficiency Evaluation for CODL

We evaluate the efficiency improvement of utilizing local hierarchical reclustering optimizations in Section IV. We compare our fully optimized algorithm CODL with two baselines CODL$^-$, which obtains the community hierarchy $\mathcal{H}_\ell(q)$ by the LORE algorithm but uses compressed COD evaluation without a HIMOR-index, and CODR.

Fig. 9 shows the execution times of three methods across six datasets. The fully optimized CODL is significantly more efficient than the two baselines. On DBLP, CODL takes 7.9 seconds on average for COD processing, whereas CODR requires 196.34 seconds. Thus, CODL is about 25$\times$ faster than CODR. CODL runs faster than CODL$^-$ at least 5 times and up to 10 times on larger datasets (Amazon and DBLP) since by utilizing the HIMOR-index, CODL only performs compressed COD evaluation over small communities in $\mathcal{I}_q$, whereas CODL$^-$ performs compressed COD evaluation over all hierarchical communities from $\mathcal{H}_\ell(q)$. CODL$^-$ is faster than CODR since CODR performs hierarchical clustering over g_ℓ from scratch, whereas CODL$^-$ only needs to recluster the community C_ℓ to generate the hierarchical community set $\mathcal{H}_\ell(q)$. The results reveal that LORE not only finds better characteristic communities but also improves efficiency.

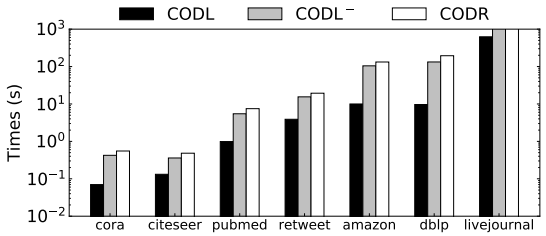


Fig. 9. Runtime for local hierarchical reclustering.

TABLE II
TIME AND MEMORY OVERHEAD OF HIMOR-INDEX AND INPUT DATA.

Dataset	Time(s)	Index Size(MB)	Input Size(MB)
Cora	0.81	0.15	0.12
CiteSeer	0.53	0.12	0.10
PubMed	16.18	1.29	1.01
Retweet	113.42	1.31	1.0
Amazon	317.63	24.55	17.38
DBLP	441.23	22.40	18.11
LiveJournal	6034.58	609.89	295.61

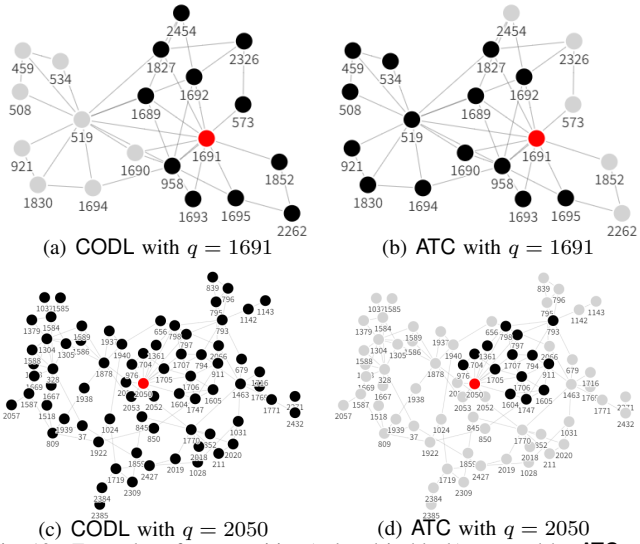
Index Overhead. Table II presents the index construction time and memory cost for COD processing and the memory consumption of the input data (including the data graph and the community hierarchy) across all datasets. The result shows that the HIMOR-index is constructed efficiently and only leads to memory overhead comparable to the input data for COD processing. Specifically, on Amazon and DBLP with over 300k nodes, it takes less than 10 minutes for index construction while the memory overhead is less than 25MB. In addition, we observe that the HIMOR-index construction on Retweet is more time-consuming than that on PubMed with a similar size. According to Section IV-B, the time complexity of index construction is linear with $\sum_{v\in\mathcal{V}_g}$ **dep**(v), which reflects the balancedness of $\mathcal{T}$. In Table I, we observe that the average depth $|\bar{\mathcal{H}}_\ell(q)|$ of Retweet is 165.3, which is an order of magnitude larger than $\log_2|\mathcal{V}_g| = 14.17$. Thus, the tree structure of communities $\mathcal{T}$ on Retweet is severely skewed, which incurs a larger construction overhead. Finding balanced hierarchical communities is orthogonal to our work. Interested readers may refer to [60] for related work. Nevertheless, it is possible to integrate any balanced hierarchical clustering method into our framework.

Scalability. We further conduct experiments on the largest dataset LiveJournal to confirm the scalability of CODL. The execution times of different methods are presented in Fig. 9. The fully optimized method CODL can handle queries from LiveJournal within 624.4 seconds by employing the HIMOR-index of memory overhead 609.89MB, whereas both COD$^-$ and CODR cannot process queries on LiveJournal within the time limit of 1000 seconds.

E. Case Study on Cora

In this section, we validate that CODL provides characteristic communities that are different from the communities returned by traditional community search methods through a case study on Cora with $k = 1$.

Figs. 10(a) and 10(b) visualize the characteristic community produced by CODL and the community returned by



(a) CODL with $q = 1691$ (b) ATC with $q = 1691$

(c) CODL with $q = 2050$ (d) ATC with $q = 2050$

Fig. 10. Examples of communities (colored in black) returned by ATC and CODL for query node 1691 and 2050 (colored in red) on Cora.

an attributed community search method ATC for the query node 1691. Node 1691 is at the center and the most (top-1) influential node in the characteristic community returned by CODL. On the contrary, the community returned by ATC is centered around node 519. Node 1691 is ranked third and is not the most influential node in the community. Furthermore, the community identified by CODL exhibits superior quality in terms of conductance when compared to the community returned by ATC. In particular, the conductance of the characteristic community is 0.285, which is lower than the conductance of the community discovered by ATC, which is 0.76. Additionally, ACQ returns a community with 109 nodes where node 1691 is ranked 34th, and CAC returns a very small community with only six nodes $\{519, 958, 1689, 1690, 1691, 1692\}$, where node 1691 is also ranked third.

Figs. 10(c) and 10(d) present the communities discovered by CODL and ATC for query node 2050, respectively. The query node 2050 is the most (top-1) influential node in both communities. Nevertheless, the community returned by CODL is of size 71, a supergraph of the community returned by ATC of size 15. Thus, when the query node holds the same influence level over the communities, CODL discovers a larger characteristic community than traditional community search methods. The conductance of the characteristic community returned by CODL, which equals to 0.23, is still lower than the conductance of the community returned by ATC, which is 0.247. Additionally, ACQ returns a community of size 43, where node 2050 is ranked second; CAC still returns a small community of four nodes $\{797, 798, 1707, 2050\}$, in which query node 2050 ranked first.

In summary, CODL discovers communities within which the query node holds equal or higher impacts than in communities discovered by traditional community search methods.

VI. Conclusion and Future Work

We introduced a novel problem of *characteristic community discovery* (COD), which seeks the largest community in which the query node has a high impact, using any given hierarchy of communities, and proposed a compressed COD evaluation framework. Furthermore, we crafted the LORE algorithm to address the COD problem over attributed graphs, which eschews the skewness of hierarchical communities when taking into account node attributes. Finally, we accelerated COD query processing with a HIMOR index. Extensive experimental results on real-world networks confirmed the effectiveness and efficiency of our proposal.

For future work, community search problems over heterogeneous information networks (HINs) have been considered recently [61]–[63]. Finding a community hierarchy for COD with multiple node and edge types and evaluating the influences of nodes in different contexts impose substantial technical challenges. Thus, it would be interesting to extend the COD problem to HINs. Furthermore, another future direction involves adapting the HIMOR-Index for dynamic graphs.

References

[1] X. Huang and L. V. S. Lakshmanan, "Attribute-driven community search," *Proc. VLDB Endow.*, vol. 10, no. 9, pp. 949–960, 2017.

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

[3] Y. Zhu, J. He, J. Ye, L. Qin, X. Huang, and J. X. Yu, "When structure meets keywords: Cohesive attributed community search," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020, pp. 1913–1922.

[4] L. Chen, C. Liu, K. Liao, J. Li, and R. Zhou, "Contextual community search over large social networks," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, 2019, pp. 88–99.

[5] Q. Liu, Y. Zhu, M. Zhao, X. Huang, J. Xu, and Y. Gao, "VAC: Vertex-centric attributed community search," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, 2020, pp. 937–948.

[6] R.-H. Li, L. Qin, J. X. Yu, and R. Mao, "Influential community search in large networks," *Proc. VLDB Endow.*, vol. 8, no. 5, pp. 509–520, 2015.

[7] J. Li, X. Wang, K. Deng, X. Yang, T. Sellis, and J. X. Yu, "Most influential community search over large social networks," in *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, 2017, pp. 871–882.

[8] W. Luo, X. Zhou, K. Li, Y. Gao, and K. Li, "Efficient influential community search in large uncertain graphs," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 4, pp. 3779–3793, 2023.

[9] D. McKenzie-Mohr, *Fostering Sustainable Behavior: An Introduction to Community-based Social Marketing*. New Society Publishers, 2011.

[10] C. Cooper, "Successfully changing individual travel behavior: Applying community-based social marketing to travel choice," *Transportation Research Record*, vol. 2021, no. 1, pp. 89–99, 2007.

[11] D. Mckenzie-Mohr, "New ways to promote proenvironmental behavior: Promoting sustainable behavior: An introduction to community-based social marketing," *Journal of Social Issues*, vol. 56, no. 3, pp. 543–554, 2000.

[12] R. Davis, I. Piven, and M. Breazeale, "Conceptualizing the brand in social media community: The five sources model," *Journal of Retailing and Consumer Services*, vol. 21, no. 4, pp. 468–481, 2014.

[13] J. H. Plikas, D. K. Nasiopoulos, D. P. Sakas, and D. S. Vlachos, "Modeling the promotion process of academic conferences through social media," in *Strategic Innovative Marketing: 4th IC-SIM, Mykonos, Greece 2015*, 2017, pp. 499–505.

[14] I. Thomson and R. Brain, "A primer in community-based social marketing," *All Current Publications*, p. 1639, 2016.

[15] M. E. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Phys. Rev. E*, vol. 69, no. 2, p. 026113, 2004.

[16] L. Dhulipala, D. Eisenstat, J. Łącki, V. Mirrokni, and J. Shi, "Hierarchical agglomerative graph clustering in nearly-linear time," in *Proceedings of the 38th International Conference on Machine Learning*, 2021, pp. 2676–2686.

[17] S. Yu, Y. Wang, Y. Gu, L. Dhulipala, and J. Shun, "ParChain: A framework for parallel hierarchical agglomerative clustering using nearest-neighbor chain," *Proc. VLDB Endow.*, vol. 15, no. 2, pp. 285–298, 2021.

[18] A. Agarwal, S. Khanna, H. Li, and P. Patil, "Sublinear algorithms for hierarchical clustering," *Adv. Neural Inf. Process. Syst.*, vol. 35, pp. 3417–3430, 2022.

[19] L. Dhulipala, D. Eisenstat, J. Lacki, V. Mirrokni, and J. Shi, "Hierarchical agglomerative graph clustering in poly-logarithmic depth," *Adv. Neural Inf. Process. Syst.*, vol. 35, pp. 22 925–22 940, 2022.

[20] M. Kapralov, A. Kumar, S. Lattanzi, and A. Mousavifar, "Learning hierarchical cluster structure of graphs in sublinear time," in *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms*, 2023, pp. 925–939.

[21] C. Borgs, M. Brautbar, J. Chayes, and B. Lucier, "Maximizing social influence in nearly optimal time," in *Proceedings of the Twenty-fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2014, pp. 946–957.

[22] Y. Tang, Y. Shi, and X. Xiao, "Influence maximization in near-linear time: A martingale approach," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, 2015, pp. 1539–1554.

[23] H. T. Nguyen, M. T. Thai, and T. N. Dinh, "Stop-and-stare: Optimal sampling algorithms for viral marketing in billion-scale networks," in *Proceedings of the 2016 International Conference on Management of Data*, 2016, pp. 695–710.

[24] K. Huang, S. Wang, G. Bevilacqua, X. Xiao, and L. V. S. Lakshmanan, "Revisiting the stop-and-stare algorithms for influence maximization," *Proc. VLDB Endow.*, vol. 10, no. 9, pp. 913–924, 2017.

[25] C. Bothorel, J. D. Cruz, M. Magnani, and B. Micenkova, "Clustering attributed graphs: Models, measures and methods," *Netw. Sci.*, vol. 3, no. 3, pp. 408–444, 2015.

[26] Y. Niu, Y. Li, J. Fan, and Z. Bao, "Local clustering over labeled graphs: An index-free approach," in *2022 IEEE 38th International Conference on Data Engineering (ICDE)*, 2022, pp. 2805–2817.

[27] Y. Meng, J. Huang, G. Wang, C. Zhang, H. Zhuang, L. Kaplan, and J. Han, "Spherical text embedding," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[28] T. Kenter and M. De Rijke, "Short text similarity with word embeddings," in *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, 2015, pp. 1411–1420.

[29] H. Duan, Y. Yang, and K. Y. Tam, "Learning numeracy: A simple yet effective number embedding approach using knowledge graph," in *Findings of the Association for Computational Linguistics: EMNLP 2021*, 2021, pp. 2597–2602.

[30] Y. Li, W. Chen, Y. Wang, and Z.-L. Zhang, "Influence diffusion dynamics and influence maximization in social networks with friend and foe relationships," in *Proceedings of the Sixth ACM International Conference on Web Search and Data Mining*, 2013, pp. 657–666.

[31] W. Lu, F. Bonchi, A. Goyal, and L. V. S. Lakshmanan, "The bang for the buck: Fair competitive viral marketing from the host perspective," in *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2013, pp. 928–936.

[32] W. Chen, W. Lu, and N. Zhang, "Time-critical influence maximization in social networks with time-delayed diffusion process," in *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012, pp. 592–598.

[33] J. Goldenberg, B. Libai, and E. Muller, "Talk of the network: A complex systems look at the underlying process of word-of-mouth," *Mark. Lett.*, vol. 12, no. 3, pp. 211–223, 2001.

[34] M. Granovetter, "Threshold models of collective behavior," *Am. J. Sociol.*, vol. 83, no. 6, pp. 1420–1443, 1978.

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

[36] Y. Li, J. Fan, Y. Wang, and K.-L. Tan, "Influence maximization on social graphs: A survey," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 10, pp. 1852–1872, 2018.

[37] W. Chen, C. Wang, and Y. Wang, "Scalable influence maximization for prevalent viral marketing in large-scale social networks," in *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2010, pp. 1029–1038.

[38] W. Chen, Y. Wang, and S. Yang, "Efficient influence maximization in social networks," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2009, pp. 199–208.

[39] 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.

[40] Y. Wu, R. Jin, J. Li, and X. Zhang, "Robust local community detection: On free rider effect and its elimination," *Proc. VLDB Endow.*, vol. 8, no. 7, pp. 798–809, 2015.

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

[42] J. Xu, X. Fu, Y. Wu, M. Luo, M. Xu, and N. Zheng, "Personalized top-n influential community search over large social networks," *World Wide Web*, vol. 23, no. 3, pp. 2153–2184, 2020.

[43] M. Charikar and V. Chatziafratis, "Approximate hierarchical clustering via sparsest cut and spreading metrics," in *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2017, pp. 841–854.

[44] M. Charikar, V. Chatziafratis, and R. Niazadeh, "Hierarchical clustering better than average-linkage," in *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2019, pp. 2291–2304.

[45] B. Moseley and J. Wang, "Approximation bounds for hierarchical clustering: Average linkage, bisecting k-means, and local search," *Adv. Neural Inf. Process. Syst.*, vol. 30, pp. 3094–3103, 2017.

[46] M. Cochez and H. Mou, "Twister tries: Approximate hierarchical agglomerative clustering for average distance in linear time," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, 2015, pp. 505–517.

[47] A. Abboud, V. Cohen-Addad, and H. Houdrougé, "Subquadratic high-dimensional hierarchical clustering," *Adv. Neural Inf. Process. Syst.*, vol. 32, pp. 11 576–11 586, 2019.

[48] M. A. Bender, M. Farach-Colton, G. Pemmasani, S. Skiena, and P. Sumazin, "Lowest common ancestors in trees and directed acyclic graphs," *J. Algorithms*, vol. 57, no. 2, pp. 75–94, 2005.

[49] S. Nassar, J. Sander, and C. Cheng, "Incremental and effective data summarization for dynamic hierarchical clustering," in *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, 2004, pp. 467–478.

[50] Q. Tu, J. Lu, B. Yuan, J. Tang, and J.-Y. Yang, "Density-based hierarchical clustering for streaming data," *Pattern Recognition Letters*, vol. 33, no. 5, pp. 641–645, 2012.

[51] G. Tong, W. Wu, S. Tang, and D.-Z. Du, "Adaptive influence maximization in dynamic social networks," *IEEE/ACM Transactions on Networking*, vol. 25, no. 1, pp. 112–125, 2016.

[52] B. Peng, "Dynamic influence maximization," *Advances in Neural Information Processing Systems*, vol. 34, pp. 10 718–10 731, 2021.

[53] H. Zhuang, Y. Sun, J. Tang, J. Zhang, and X. Sun, "Influence maximization in dynamic social networks," in *2013 IEEE 13th International Conference on Data Mining*, 2013, pp. 1313–1318.

[54] I. Gronau and S. Moran, "Optimal implementations of upgma and other common clustering algorithms," *Inf. Process. Lett.*, vol. 104, no. 6, pp. 205–210, 2007.

[55] F. Murtagh, "A survey of recent advances in hierarchical clustering algorithms," *Computer J.*, vol. 26, no. 4, pp. 354–359, 1983.

[56] A. Goyal, F. Bonchi, and L. V. S. Lakshmanan, "A data-based approach to social influence maximization," *Proc. VLDB Endow.*, vol. 5, no. 1, pp. 73–84, 2011.

[57] P. Ronhovde and Z. Nussinov, "Multiresolution community detection for megascale networks by information-based replica correlations," *Phys. Rev. E*, vol. 80, no. 1, p. 016109, 2009.

[58] ——, "Local resolution-limit-free potts model for community detection," *Phys. Rev. E*, vol. 81, no. 4, p. 046114, 2010.

[59] R. K. Ronhovde, D. R. Reichman, P. Ronhovde, and Z. Nussinov, "An edge density definition of overlapping and weighted graph communities," *arXiv:1301.3120*, 2013.

[60] M. Knittel, M. Springer, J. P. Dickerson, and M. Hajiaghayi, "Generalized reductions: Making any hierarchical clustering fair and balanced with low cost," in *Proceedings of the 40th International Conference on Machine Learning*, 2023, pp. 17 218–17 242.

[61] X. Jian, Y. Wang, and L. Chen, "Effective and efficient relational community detection and search in large dynamic heterogeneous information networks," *Proc. VLDB Endow.*, vol. 13, no. 10, pp. 1723–1736, 2020.

[62] Y. Jiang, Y. Fang, C. Ma, X. Cao, and C. Li, "Effective community search over large star-schema heterogeneous information networks," *Proc. VLDB Endow.*, vol. 15, no. 11, pp. 2307–2320, 2022.
[63] Y. Fang, Y. Yang, W. Zhang, X. Lin, and X. Cao, "Effective and efficient community search over large heterogeneous information networks," *Proc. VLDB Endow.*, vol. 13, no. 6, pp. 854–867, 2020.