Discover Your Characteristic Community from Graph Hierarchical Communities

Niu Yudong
ydniu.2018@phdcs.smu.edu.sg
Singapore Management University
Singapore

Yuchen Li
yuchenli@smu.edu.sg
Singapore Management University
Singapore

Panagiotis Karras
piekarras@gmail.com
Aarhus University
Aarhus, Denmark

ABSTRACT

What is the widest community in which a person exercises strong influence? This question arises in many real-world circumstances such as online social marketing and event organization. While much attention has been devoted to finding the most influential set of members in a given community, the problem of finding the most characteristic community of influence for a given person has barely been examined. In this paper, we study this novel problem of *characteristic community discovery* (COD): given a query node within a set of hierarchical communities, find the largest community in which that node is impactful. The key challenge of the COD problem is to verify the impact of the query node in each community. We first propose the compressed COD framework to accelerate the evaluation of impact in communities by eliminating the redundant computations caused by overlap among communities. We then study the COD problem over labeled graphs. To efficiently take query labels into consideration, we propose *locally hierarchical reclustering*, which avoids globally reclustering the graph for different query labels. Lastly, we propose an efficient index approach to avoid impact verification of communities that are not reclustered. Extensive experiments on networks with both real and synthetic labels validate the effectiveness and efficiency of our solutions to COD. Our hierarchical method finds better characteristic communities than traditional community search methods and achieves up to 25 times acceleration over baselines.

PVLDB Reference Format:
Niu Yudong, Yuchen Li, and Panagiotis Karras. Discover Your Characteristic Community from Graph Hierarchical Communities. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/YudongNiu/COD.

1 INTRODUCTION

Graphs are the prevailing model to represent social ties among persons using nodes and edges. Large social graphs often exhibit a complex, multi-scale community structures where a node belongs to many node clusters — *communities* — of diverse sizes [1, 23, 36]. For instance, researchers within an area of study form a community through collaborations. Further, a researcher studying the problem of *query rewrite* belongs in increasingly larger communities of broader topics, such as *query optimization* and *data management.* To understand the influence structure of such communities and make informed decisions in policy-making, science, and business, one needs to recognize the largest community in which a chosen individual has impact.

For example, advertisers utilizing social media platforms like *Meta* or *Twitter* aim to have campaigns promoted by *key opinion leaders* (KOLs), who may exercise strong impact over their communities; to do so, an advertiser should be able to recognize the largest community over which a KOL has impact. Similarly, an *event organizer* utilizing an online social platform such as *Meetup* or *EventBrite* aims to utilize resources by inviting the largest group of participants that are more likely to accept the invite, based on the connections and shared interests among themselves and the organizer. Besides, one may use knowledge of largest community over which a certain individual exercises strong influence as an community-based *outstanding fact* (OF) about that person; that is, a striking claim that makes one stand out from their peers [39] and could be used in data journalism and fact checking.

To address these needs, in this paper we propose the novel problem of *C̲haracteristic cO̲mmunity D̲iscovery* (COD). Given a query node q in a graph G with hierarchical communities, find the *characteristic community* of q, i.e., the largest community $C^*(q)$ in which q is a top-k influential node.

The COD problem is computationally challenging; it requires evaluating *all* relevant hierarchical communities, since the impact of a node in its hierarchical communities is *non-monotonic*: a node that is not impactful in a smaller-scale community may still be impactful in larger-scale communities. For instance, a researcher in an interdisciplinary area may not be impactful in any specialization community. However, the same researcher may be impactful in the combined community by virtue of playing a bridge role. A straightforward approach to COD would evaluate the community impact of the query node in each community independently. Unfortunately, even equipped with the state-of-the-art sampling-based solutions on finding influential nodes [18, 31, 38], this straightforward approach is prohibitively expensive, as there could be hundreds of relevant hierarchical communities to a query node on large graphs.

To address this need, we propose a compressed COD framework that efficiently discovers the characteristic community. Grounded on the observation that the communities at deeper levels of the hierarchy are contained in those at shallower levels, we eliminate redundant computations in two steps. First, we generate samples from the original graph and *share* these samples for influence evaluation on communities at different scales. Second, we develop an *incremental update* method that avoids finding the top-k influential

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

nodes for each community from scratch; instead, it progressively maintains the top-k nodes from deeper to shallower levels. Theoretically, the time complexity of the compressed COD framework is $O(\Theta \cdot \omega + |\mathcal{H}(q)|)$, which completely decouples the sampling cost on the original graph ($\Theta \cdot \omega$) from the number of hierarchical communities containing the query node ($|\mathcal{H}(q)|$).

We further study COD over labeled graphs. In many real-world social networks, nodes are annotated with labels capturing their properties. For example, labels can indicate research areas in a collaboration network. Such labels need to be taken in consideration in COD. If one wants to organize an academic conference on certain research areas, a characteristic community that comprises researchers from matched areas is a preferable set for the organizer to send invites.

Over a labeled graph, the COD problem receives an additional query label as input and aims at revealing a characteristic community related to the query label. The key challenge is then to properly update the community hierarchy according to the query label. A direct approach would transform the original graph to a weighted graph by setting higher weights to edges connecting nodes with the query's label, and obtain a label-aware community hierarchy by feeding this weighted graph to a hierarchical clustering algorithm. Unfortunately, this direct approach has two major drawbacks: *first*, it is costly to perform hierarchical clustering from scratch for different query labels; *second*, label-aware transformation can cause a *free-rider effect* [40] and thus lead to suboptimal characteristic communities. As we will see in our experimental study (Fig. 6(g)–6(r)), characteristic communities discovered by the direct approach have unsatisfactory topology and label densities.

To avoid such reclustering from scratch, we propose the *LOcally hierarchical REclustering* (LORE) approach. We first construct a community hierarchy $\mathcal{T}$ on the original graph by ignoring label information. Using $\mathcal{T}$ as a reference, a novel *reclustering score* is devised to identify a proper local community $C \in \mathcal{T}$ that should be reclustered to resolve the conflicts between the unlabeled community hierarchy and the query label. We then recluster C and integrate the result hierarchy into $\mathcal{T}$. LORE also enables us to accelerate the COD query using an index. We precompute the influence ranks of nodes in $\mathcal{T}$ and avoid redundant rank re-computations for communities that are not reclustered.

Our contributions are then summarized as follows:

- We propose the novel problem of *characteristic community discovery* (COD) that studies the community impact of graph nodes (Sec. 2).
- We devise the compressed COD framework that eliminates redundant impact evaluations in hierarchical communities and scales the computation to large graphs (Sec. 3).
- We propose LORE, which supports COD over labeled graphs by avoiding reclustering from scratch; we further develop an index-based method on top of LORE (Sec. 4).
- Through extensive experiments on networks with real-world and synthetic node labels, 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 25 times speedup over baselines (Sec. 5).

2 PRELIMINARIES

In this section, we first introduce the basic notations and problem formulation of the *characteristic community discovery* (COD). Subsequently, we discuss the related literature.

2.1 Notations and Problem Statement

Graphs. Consider a connected graph $g = (\mathcal{V}_g, \mathcal{E}_g)$ with node set $\mathcal{V}_g$ and edge set $\mathcal{E}_g$. When g is a weighted graph, each edge $(u, v) \in \mathcal{E}_g$ is associated with a weight $w(u, v)$. We denote the set of neighbors of a node $v \in \mathcal{V}_g$ as $\mathcal{N}_g(v)$. For the ease of presentation, we drop the subscript g when the context is clear.

Hierarchical Communities. Given a graph g, a community C is an *induced* subgraph of g and $|C|$ denotes the number of nodes in C. The community hierarchy of g is a set of communities $\mathcal{T} = \{C_0, C_1, C_2, ...\}$ organized in a tree structure. The root vertex of $\mathcal{T}$ is a community that contains all nodes in g and each leaf vertex contains 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 $C \in \mathcal{T}$, $\mathbf{dep}(C)$ denotes the depth of C from the root vertex in $\mathcal{T}$. For a node $u \in \mathcal{V}$, a set of hierarchical communities $\mathcal{H}(u) = \{C_0(u), C_1(u), ...\}$ is obtained by extracting all communities in $\mathcal{T}$ that contain u. Further, the communities in $\mathcal{H}(u)$ are sorted based on *decreasing* order of their depths, i.e., $\mathbf{dep}(C_i(u)) > \mathbf{dep}(C_j(u))$ when $i < j$ and thus $C_{|\mathcal{H}(u)|}(u) = g$. We use $\mathcal{H}(u|C)$ to denote the subset of $\mathcal{H}(u)$ containing 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., the lowest common ancestor of u and v in $\mathcal{T}$. We also use $\mathbf{lca}(v)$ to denote the parent of v. We further abuse the notation and use $\mathbf{dep}(u, v)$ to denote $\mathbf{dep}(\mathbf{lca}(u, v))$ for ease of presentation.

Influence Models. Various influence models have been proposed to evaluate the social influence of nodes in a graph [7, 25, 27]. For simplicity, we illustrate the independent cascade (IC) model [13] as an example. Nevertheless, our proposed method can support other influence models such as the linear threshold model [15] and other triggering models [21] with simple adaptions.

Under the IC model, each edge (u, v) is associated with an *influence probability* $p(u, v)$ to measure the influence from u to v on graph 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 subsequent step t, a node u that is just activated at step $t - 1$ tries to activate each of its inactivated neighbors $v \in \mathcal{V}$ with probability $p(u, v)$. Note that each activated node has only one chance to activate its neighbors. The propagation process terminates when no more nodes can be activated. The *influence* $\sigma_g(q)$ of q in g is measured by the expected number of activated nodes over the stochastic process. We use $\mathbf{rank}_g(q)$ to denote the *influence rank* of q in g, i.e., the number of nodes that have larger influence than q.

RR Set. Computing the exact influence of a node is known as #P-hard [8]. The *reverse reachable* (RR) set is a general and efficient method to estimate the influence for a wide range of influence models [4]. A random RR set r_g in a graph g is generated as follows. At step 0, a *source* node s is uniformly sampled from $\mathcal{V}$ and added to r_g as an initial activated node. Then, at each step t, a node v that is just activated has one chance to reversely activate each of its neighbors $u \in \mathcal{V}$ that has not been added into r_g with probability

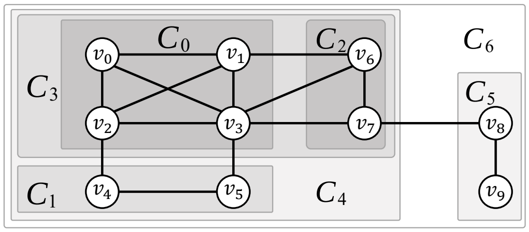


Figure 1: An example of hierarchical communities.

Table 1: Frequently used notations.

Notations	Description
$g = (\mathcal{V}_g, \mathcal{E}_g)$	Graph g with node set $\mathcal{V}_g$ and edge set $\mathcal{E}_g$.
$\mathcal{T}$	The community hierarchy of graph g organized in a tree structure.
C	A community from $\mathcal{T}$.
$\mathcal{H}(u)$	The set of communities in $\mathcal{T}$ that contains u and sorted according to the depths of these communities.
$\mathcal{H}(u\|C)$	The subset of $\mathcal{H}(u)$ containing descendants of C.
$\mathbf{dep}(C)$	The depth of C from the root vertex in $\mathcal{T}$.
$\mathbf{lca}(u, v)$	The lowest common ancestor of u and v in $\mathcal{T}$.
r_g, R_g	A random RR set and the corresponding RR graph.
$\mathbf{rank}_g(q)$	The influence rank of node q in g.
$C^*(q)$	The characteristic community of node q.

$p(u, v)$. The process terminates when no more node can be activated further. The following theorem shows that the influence can be estimated with RR sets.

Theorem 2.1 ([4]). *Given a node $q \in \mathcal{V}$, $\sigma_g(q) = p_g(q) \cdot |\mathcal{V}|$ where $p_g(q)$ represents the probability of q contained in a RR set.*

Problem Statement. In this work, we focus on characterizing the *community impact* of a node on large graphs. Specifically, we study the problem of *Characteristic cOmmunity Discovery* (COD), which aims at finding the *largest* community from hierarchical communities so that the query node is a top-k influential node in the community. We formally define the problem of COD as follows.

Definition 2.2 (COD). Given a graph g, a query node q and hierarchical communities $\mathcal{H}(q)$, we find the characteristic community $C^*(q)$ as:

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

Example 2.3. Fig. 1 shows a graph g with community hierarchy $\mathcal{T} = \{C_0, C_1, ..., C_6\}$. Given two nodes v_0 and v_6, the lowest common ancestor $\mathbf{lca}(v_0, v_6) = C_3$ which has $\mathbf{dep}(C_3) = 3$. Given the query node v_0 and $k = 1$, the hierarchical communities containing v_0 is $\mathcal{H}(v_0) = \{C_0, C_3, C_4, C_6\}$. The subset of hierarchical communities $\mathcal{H}(q|C_4) = \{C_0, C_3\}$. The influence ranks of v_0 in communities from $\mathcal{H}(v_0)$ are $\mathbf{rank}_{C_0}(v_0) = 1$, $\mathbf{rank}_{C_3}(v_0) = 3$ and $\mathbf{rank}_{C_4}(v_0) = \mathbf{rank}_{C_6}(v_0) = 2$. Thus, C_0 is returned as the characteristic community $C^*(v_0)$.

Before moving on, all frequently used notations are summarized in Table 1.

2.2 Related Work

We review related studies of the COD problem: (1) community search; (2) influence maximization; (3) graph hierarchical clustering.

Community Search. The problem aims at finding a densely connected subgraph containing one or many query nodes. Various community models have been devised based on k-core [2, 35], k-truss [10, 19] and others [6, 9, 17, 28]. To support community search over labeled graphs, [12] and [42] return the maximal k-core and triangle-connected k-truss in which each node shares the query labels as the discovered community, respectively. However, most studies focus on the structure or label cohesiveness of the result communities, and do not consider the influence of the query node q in the discovered community.

There are a few related works on community search that consider the influence of the query node indirectly. [37] proposes the constrained-core model, which finds the largest k-core such that the sum of distances from nodes in that k-core to the query node is less than a given limit. [40] finds the subgraph that contains the query node with the largest query-biased density as the discovered community. ATC [20] considers the status of the query node for community search over labeled graphs. The authors propose to find the (k, d)-truss[1] that maximizes certain query-label density function. Compare with existing works that consider the node influence indirectly, the proposed COD problem ensures that the query node is a top-k influential node in the discovered community.

Influence Maximization. The influence of nodes in networks have been widely studied in the literature. Most of existing works focus on the problem of influence maximization, which targets at finding k seed nodes with the maximum influence. A plethora of studies have been proposed to optimize the efficiency of computing the optimal seed nodes (see the survey in [26]). However, existing works on influence maximization only evaluate the influence of nodes over the entire network. In contrast, our work focus on the influence of nodes at different community scales and discover the characteristic community for a query node. In other words, even if a node is not influential over the entire network, COD finds a community where the node can made a significant impact.

Graph Hierarchical Clustering. The problem aims to recursively cluster the graph into hierarchical communities. There are two types of methods for graph hierarchical clustering. The first type of methods adopt the divisive (top-down) approach. For example, [30] partitions the graph into hierarchical communities by iteratively removing the edge with maximum betweenness centrality. The second type of methods adopt the agglomerative (bottom-up) approach [11], which initialize each node as a single community and iteratively merge two communities that maximize a linkage function to form larger communities. Obtaining the hierarchical communities is orthogonal to the COD problem as we can adopt any methods that produce the community hierarchy.

Summary. The COD problem builds a strong connection between the three areas of related works. Given any hierarchical communities, COD considers both the influence of q and community cohesiveness to discover the characteristic community of q.

[1] Given the query node q and numbers k and d, a (k, d)-truss is a connected k-truss containing q and the distance between each node in the k-truss and q is less than d.

3 COMPRESSED COD EVALUATION

Given a query node q and hierarchical communities $\mathcal{H}(q)$, q can be a top-k influential node in a larger community but not in a smaller subcommunity, as it may play a *bridge* role among smaller communities and be impactful over their union only. It follows that we need to compute the influence rank of q over all communities in $\mathcal{H}(q)$ to find the correct characteristic community $C^*(q)$.

Observation 1. *The influence rank* $\mathbf{rank}_C(q)$ *of the query node* q *is not monotonic with* $\mathbf{dep}(C)$.

Based on Observation 1, we devise a two-step framework to solve COD: we estimate the influence score $\sigma_C(v)$ of each node v in each community $C \in \mathcal{H}(q)$; then we find the top-k per community and return the largest community where q is in the top-k.

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

We thus propose the *compressed evaluation* framework that eschews redundant influence and rank computations incurred by overlaps among communities. We introduce a two-step compressed processing approach: *Shared Sample Generation* and *Incremental Top-k Evaluation*. As our complexity analysis reveals, this approach reduces redundant computations to the minimum.

3.1 Shared Sample Generation

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

Definition 3.1 (RR graph). Given an RR set r_g, the 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 $\mathcal{E}_g$ that is activated during the generation of r_g.

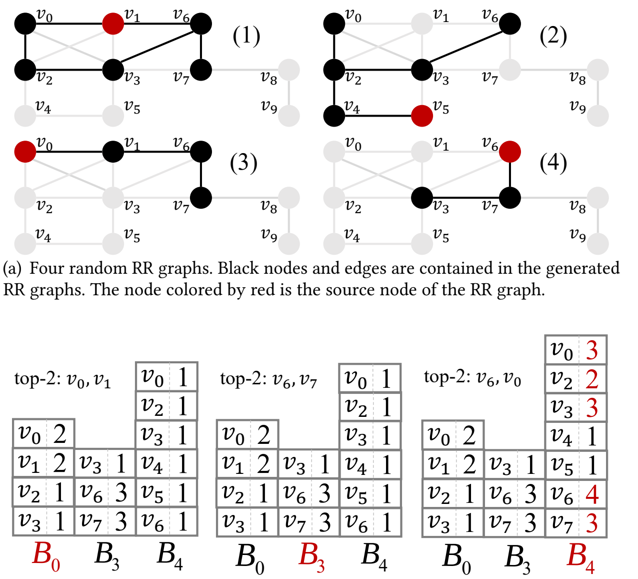
Definition 3.2 (Induced RR graph). Given a random RR graph R_g, a community C that contains the source node of R_g, we define the induced RR graph as $R_g(C) = (r_g \cap \mathcal{V}_C, \mathcal{E}_g \cap \mathcal{E}_C)$.

The following theorem shows that the induced RR graph can be used for estimating influence.

Theorem 3.3. *Given 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 an induced RR graph* $R_g(C)$.

Proof. Consider a possible world g_i on graph g where each edge (u, v) is sampled to be activated with $p(u, v)$ independently. For any source node $s \in C$, an RR set from s on C generated wrt g_i is identical to the set of reachable nodes from s by only traversing nodes in C, i.e., an induced RR graph $R_g(C)$ where R_g is generated from s wrt g_i. By taking the expectation over all possible g_i, we can deduce the theorem. □

Based on Theorem 3.3, we can generate an adequate number of random RR graphs on the original graph g to estimate the influence of nodes in different communities.



(a) Four random RR graphs. Black nodes and edges are contained in the generated RR graphs. The node colored by red is the source node of the RR graph.

(b) Buckets storing the influence estimations. The subfigure on the left side denotes the buckets produced by HFS algorithm. The following two subfigures presents buckets during and after the incremental top-k evaluation process.

Figure 2: Example of Random RR graphs and the corresponding buckets for storing influence estimations.

We propose the *hierarchical first search* (HFS) algorithm that efficiently compresses the estimation process of influence scores with induced RR graphs. For an instance of RR graph R_g, HFS starts the traversal from R_g's source node s. We first explore the *smallest* community $C \in \mathcal{H}(q)$ that contains s and uses the corresponding induced RR graph $R_g(C)$ to update node influence scores. Subsequently, HFS iteratively explores increasingly larger communities and record node influence along the way. Note that for a node v in R_g, we only record its influence score in the community where HFS *first* visits v. Although the influence scores in larger communities should also be recorded, we delay the materialization so that it allows for efficient *incremental top-k evaluation* in Section 3.2.

Example 3.4. Fig. 2(a) gives four random RR graphs. We describe the HFS algorithm on random RR graph (1) and (2) in detail. For (1), the traversal starts from the source node v_1 (the red node in Fig. 2(a)) and first traverses the induced RR graph $R_g(C_0)$. It explores nodes v_0, v_2 and $v_3 \in R_g(C_0)$ and records the node influence in bucket B_0. Then, HFS further traverses the induced RR graph $R_g(C_3)$ and explore nodes $v_6, v_7 \in R_g(C_3)$ and records the node influence in bucket B_3. For (2), the traversal starts from the source node v_5 and traverses the induced RR graph $R_g(C_4)$. It explores nodes v_4, v_2, v_0, v_3 and $v_6 \in R_g(C_4)$ and records the node influences in bucket B_4. Note that $R_g(C_0) = \varnothing$ and $R_g(C_3) = \varnothing$ for RR graph (2) since source node v_5 is not contained in C_0 and C_3. After HFS processes all four RR graphs, the first three columns of Fig. 2(b) presents the buckets obtained. The item $B_0(v_1) = 2$ since v_1 appears in the induced RR graphs over C_0 of two RR graphs (RR graph (1) and (3)). Note that v_3 appears in the induced RR graphs over C_4 of three RR graphs (RR graph (1), (2) and (3)). To recover that information from the buckets for estimating $\sigma_{C_4}(v_3)$, we need $B_0(v_3) + B_3(v_3) + B_4(v_3) = 3$.

3.2 Incremental Top-k Evaluation

Given the influences scores obtained by HFS, the second step is to check if q is a top-k influential node in the communities from $\mathcal{H}(q)$. We propose an incremental evaluation algorithm that only requires a *one-pass* processing of the buckets to obtain the top-k nodes for all communities.

The algorithm computes the top-k influential nodes for each community in $\mathcal{H}(q)$ through scanning the corresponding bucket. It first scans bucket B_0 that corresponds to the smallest community in $\mathcal{H}(q)$ and records the top-k nodes with largest values in B_0. The algorithm then iteratively moves to the next bucket that corresponds to a larger community. However, instead of computing the top-k from scratch, we use the top-k from the previous bucket and only scan the items in the current bucket to update the top-k. Theorem 3.5 ensures the correctness of the incremental approach. If the node does not appear in the current scanned bucket and also not contained in the previous top-k nodes, it cannot be the top-k influential nodes in the current bucket.

Theorem 3.5. *Given parameter k and node v, if $\mathbf{rank}_{C_h}(v) > k$ and $B_{h+1}(v)$ does not exists, then $\mathbf{rank}_{C_{h+1}}(v) > k$.*

Proof. When the samples are fixed, $\mathbf{rank}_{C_h(q)}(v) > k$ is equivalent to the existence of k nodes u_0 to $u_{k-1} \in C_h(q)$ such that 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 exists, 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, we have $\mathbf{rank}_{C_{h+1}(q)}(v) > k$. □

Example 3.6. We continue the running example in Fig. 2(b). Given query node v_0, parameter $k = 2$, the incremental algorithm first scans the buckets B_0 (the first column in Fig. 2(b)) and record the top-2 nodes $\{(v_0, 2), (v_1, 2)\}$. Next, bucket B_3 is scanned (the middle subfigure in Fig. 2(b)). $B_3(v_6) = 3$ and $B_3(v_7) = 3$ are larger than the k-th value recorded in the previous top-k nodes. Thus, the recorded top-k nodes for bucket B_3 are updated to $\{(v_6, 3), (v_7, 3)\}$. Finally, bucket B_4 is scanned (the right subfigure in Fig. 2(b)). $B_4(v_0)$ is updated as $B_4(v_0) + B_0(v_0) = 3$. Other values in B_4 are updated similarly during the scan and the recorded top-k nodes are also updated to $\{(v_6, 3), (v_0, 3)\}$. Although node v_1 exists in bucket B_0, the algorithm does not need to consider node v_1 while scanning bucket B_4. Node v_1 is neither contained in bucket B_4 nor the recorded top-k nodes for bucket B_3, thus v_1 cannot be top-k nodes in bucket B_4 according to Theorem 3.5.

3.3 Implementation and Complexities

To put things together, we present the pseudo-code of the *compressed COD framework* in Alg. 1. It takes graph g, hierarchical communities $\mathcal{H}(q)$, a query node q and the parameter k as inputs, and outputs the characteristic community $C^*(q)$ for the query node q. The algorithm first generates Θ RR graphs for influence estimation. All RR graphs are stored in the collection $\mathcal{R}$ and the source node for each RR graph is added into the priority queue Q (Line 3-7). Note that the priority queue only has $|\mathcal{H}(q)|$ distinct priority values and thus we maintain $|\mathcal{H}(q)|$ hash maps so that each insertion takes $O(1)$ time. HFS is then performed over all RR graphs and updates the buckets for each community in $\mathcal{H}(q)$ (Line 8-16). Then, the incremental top-k evaluation is performed on each bucket (Line 19-28). The heap H is used to record the top-k nodes for each bucket. The algorithm updates items in bucket B_h and add $v \in B_h$ into H if the updated $B_h(v)$ is larger than the number of activations of the current top of H (Line 20-26). If q is within H after scanning of the bucket, $C^*(q)$ is updated to the corresponding community (Line 27-28). Finally, $C^*(q)$ is returned.

Algorithm 1: Compressed COD Framework

```
   Input: graph $g$, hierarchical communities $\mathcal{H}(q)$, query node $q$,
          parameter $k$
   Output: characteristic community $C^*(q)$
   // Shared Sample Generation
1  $\mathcal{R} \leftarrow \emptyset$;
2  $Q \leftarrow$ an empty priority queue;
3  for $i$ in $1$ to $\Theta$ do
4      generate a random RR graph $R$ with source $v_s \in \mathcal{V}$;
5      $h \leftarrow |\mathcal{H}(q)| - \mathbf{dep}(v_s)$;
6      add $(v_s, R)$ into $Q[h]$;
7      $\mathcal{R} \leftarrow \mathcal{R} \cup \{R\}$;
8  for $h$ in 0 to $|\mathcal{H}(q)| - 1$ do
9      $B_h \leftarrow \emptyset$ ;
10     while $Q[h] \neq \emptyset$ do
11         $(v, R) \leftarrow Q[h]$.next();
12         update $B_h(v)$;
13         for $u$ in $\mathcal{N}_R(v)$ do
14             if $u$ is not explored then
15                 $h' \leftarrow |\mathcal{H}(q)| - \min(\mathbf{dep}(C_h), \mathbf{dep}(u))$;
16                 add $(u, R)$ to $Q[h']$;
   // Incremental Top-k Evaluation
17 $\tau(v) \leftarrow 0$ for $v \in \mathcal{V}$;
18 $H \leftarrow$ an empty min-heap of size $k$;
19 for $h$ in 0 to $|\mathcal{H}| - 1$ do
20     for $v \in B_h$ do
21         $B_h(v) \leftarrow B_h(v) + \tau(v)$;
22         $\tau(v) \leftarrow B_h(v)$;
23         if $\tau(H.\text{top}()) < B_h(v)$ and $v \notin H$ then
24             add $v$ into $H$;
25             if size$(H) > k$ then
26                 $H$.pop();
27     if $q \in H$ then
28         $C^*(q) \leftarrow C_h(q)$;
29 return $C^*(q)$;
```

To analyze the complexity of the compressed COD algorithm, we first show the following result to bound the size of buckets.

Theorem 3.7. *The value $\sum_{0 \le h < |\mathcal{H}(q)|} |B_h|$ is bounded by $O(|\mathcal{R}|)$, where $|\mathcal{R}|$ denote the number of nodes in $\mathcal{R}$.*

Proof. In the shared sample generation stage of Alg. 1, items are added into buckets when a node in $\mathcal{R}$ is explored. Note that HFS only explores a node in $\mathcal{R}$ once. Thus, the total size of the buckets are bounded by $O(|\mathcal{R}|)$ during the first stage. In the incremental top-k evaluation stage, the algorithm only accesses existing items in buckets and does not add items into buckets. Thus, the total size of the buckets during Alg. 1 is bounded by $O(|\mathcal{R}|)$. □

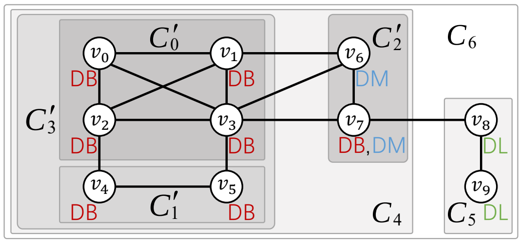


Figure 3: The graph g augmented with node labels. Given the query label ℓ_q = {DB} and query node v_0, Community C_4 in Fig. 1 is reclustered according to the LORE algorithm.

Time complexity. A collection $\mathcal{R}$ of Θ RR graphs are generated, which takes $O(\Theta \cdot \omega)$ operations and ω is the expected cost to generate a RR graph on network g. Each node in $\mathcal{R}$ is explored only once during the traversal. Thus, this step requires $O(\text{vol}(\mathcal{R}) + |\mathcal{R}|)$ operations, where $\text{vol}(\mathcal{R})$ denote the number of edges in $\mathcal{R}$. Then, in the incremental top-k evaluation stage, each bucket is scanned to obtain the top-k influential nodes in each bucket. According to Theorem 3.7, the number of operations of this step is bounded by $O(|\mathcal{R}| + |\mathcal{H}(q)|)$. Note that an additional $|\mathcal{H}(q)|$ operations are needed since the algorithm have to check buckets for each $C \in \mathcal{H}(q)$ even if the bucket is empty. Since $\Theta \cdot \omega \geq \Theta \cdot |\mathcal{E}_{R_g}| \geq |\mathcal{R}|$, the time complexity of Alg. 1 is bounded by $O(\Theta \cdot \omega + |\mathcal{H}(q)|)$.

Discussion. The analysis shows that our compressed COD framework is effective in eliminating the redundant computations among communities. Even though we need to check if the query q is in the top-k for each community from $\mathcal{H}(q)$, the term $|\mathcal{H}(q)|$ is completely decoupled from the sampling cost in the complexity results.

4 COD OVER LABELED GRAPH

In many real-world social networks, nodes are attached with labels describing their properties. Previous works [12, 20, 41] have shown the significance of considering node labels for community analysis. In this section, we study the COD problem over labeled graphs.

Label-aware COD. The COD problem over labeled graphs will take an additional input ℓ_q, which denotes the query label, and output a characteristic community $C^*(q)$. The discovered community should contain more nodes with label ℓ_q. For example, an interdisciplinary researcher can reside in multiple communities across different research domains. By incorporating the label information, the research can discover her characteristic community formed with researchers on a targeted domain specified by the query label.

To our best knowledge, there is *no* existing works on constructing hierarchical communities on labeled graphs. To support labeled graphs, we can devise the following *global reclustering* approach for a COD query:

- Adopt the existing works on label-aware community analysis [5, 41] to transform the original graph g by setting edge weights to reflect the topology and label information in a weighted graph g_ℓ;
- Recluster g_ℓ into hierarchical communities and use the compressed COD framework in Sec. 3 to get $C^*(q)$.

However, the global reclustering approach has the following drawbacks in terms of both effectiveness and efficiency.

First, it suffers from the *free-rider effect* [40]. There are two types of free-riders: (1) nodes with the query label but have low structure relevance with the query node; (2) nodes without the query labels but are structurally close to the nodes with the query label. We found that the global reclustering approach are more likely to produce characteristic communities with free-riders. As shown in Fig. 6(g)- Fig. 6(r) in the experiments, the global approach (named as CODR) is likely to produce characteristic communities with lower structure and label density due to the free-riders.

Second, the reclustering is costly if executed from scratch. Each different query label generates a different community hierarchy. Even the most efficient hierarchical clustering algorithm [11] has the time complexity $\tilde{O}(|\mathcal{V}_g| \cdot \sqrt{|\mathcal{E}_g|})$, which can be very expensive for query processing over large or dense graphs.

Thus, we propose a local reclustering approach that overcomes the deficiencies of the global approach. Furthermore, we develop an index-based method that further accelerate the query processing.

4.1 Local Hierarchical Reclustering

To mitigate the deficiencies of the global approach, we propose the *LOcal hierarchical REclustering* (LORE) algorithm that partially reclusters the graph for updating the community hierarchy. Specifically, the LORE algorithm first identifies a proper subgraph C_ℓ containing query node q from the unlabeled hierarchical communities $\mathcal{H}(q)$ (which is obtained by ignoring the label) and then reclusters C_ℓ with the global reclustering approach. The reclustered communities substitute for C_ℓ and its descendants in $\mathcal{H}(q)$ to obtain the *label-aware* hierarchical communities $\mathcal{H}_\ell(q)$.

LORE addresses the deficiencies of the global approach. First, C_ℓ is often much smaller than the entire graph g_ℓ. Thus, by only reclustering the community C_ℓ, LORE can avoid the cost for reclustering from scratch. Second, LORE relieves the free-rider effects since nodes with low structure relevance with the query node (outside community C_ℓ) are not reclustered and thus are not added into C_ℓ even with query labels. As shown in our experiments, LORE discovers characteristics communities with better structure quality and higher label density compared with the global approach.

The crux of LORE is how to choose the proper community C_ℓ to be reclustered. We design a novel *reclustering score* $r(C)$ for each community $C \in \mathcal{H}(q)$ to measure the extent that C needs to be reclustered according to ℓ_q, and the community with maximum *reclustering score* is selected as C_ℓ, i.e. $C_\ell = \arg\max_{r(C)} C$. Intuitively, a community C needs to be reclustered according to query labels if the nodes with query labels in C are badly clustered by unlabeled hierarchical clustering, i.e., a large number of edges having both ends with the query label are cut by the descendants of C in the unlabeled $\mathcal{H}(q)$. Hence, we design the reclustering score to be the ratio of the cut as the following:

Definition 4.1. Given query labels ℓ_q, unlabeled hierarchical communities $\mathcal{H}(q)$ and a community $C \in \mathcal{H}(q)$, the *reclustering score* of C is defined as:

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

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

The numerator of $r(C)$ is a summation over query-labeled edges that are cut by the boundary of descendants of C in unlabeled $\mathcal{H}(q)$. Instead of simply counting the number of such query-labeled edges, we sum $\mathbf{dep}(u, v)$ for each cut query-labeled edge (u, v). This summation is used to measure the degree of conflicts between the subset of hierarchical communities $\mathcal{H}(q|C)$ and query labels. If a query-labeled edge is cut at deeper levels of $\mathcal{H}(q|C)$, it causes higher conflicts between $\mathcal{H}(q|C)$ and the query label since the nodes with the query label around q are badly clustered. By comparison, if a query-labeled edge is cut at shallower levels of $\mathcal{H}(q|C)$, it is acceptable since the nodes of the edge are only loosely related to q in a bigger community.

Example 4.2. Fig. 3 displays the graph g in Fig. 1 augmented with node labels. Given query label ℓ_q ={DB} and 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 cut by the boundary of C_0 and $(v_2, v_4), (v_3, v_5)$ is cut by the boundary of C_3. The reclustering score of community C_4 in Fig. 1 is $r(C_4) = \frac{\sum_{(u,v)\in\delta(v_0,C_4)} \mathbf{dep}(u,v)}{|C_4|} = \frac{2+2+3}{8} = \frac{7}{8}$.

Efficient Score Computation. Note that the *reclustering score* can be computed efficiently for all $C \in \mathcal{H}(q)$ in $O(|\mathcal{E}_g|)$ times through a dynamic programming approach. We first compute the number of query-labeled edges $\Delta(C)$ that cut the boundary of $C \in \mathcal{H}(q)$ by visiting each query-labeled edge and find its lowest common ancestor. Then, we can use the recursion $r(C) \cdot |C| = r(C') \cdot |C'| + \Delta(C') \cdot \mathbf{dep}(C')$ where C' is the child of C to compute the *reclustering score* of each community.

The pseudocode of LORE is presented in Alg. 2. It takes graph g, a query with query node q and query label ℓ_q, and the community hierarchy $\mathcal{T}$ generated through unlabeled hierarchical clustering over g as inputs, and output hierarchical communities $\mathcal{H}_\ell(q)$ that are related to the query label. The LORE algorithm first identifies the community C_ℓ that needs to be reclustered using Function QueryLabelRelated (Line 1) and then reclusters C_ℓ through hierarchical clustering according to weights in g_ℓ (Line 2 to 3). Finally, the algorithm returns the union of label-aware communities within C_ℓ and the ancestors of C_ℓ in the unlabeled community hierarchy $\mathcal{T}$ (Line 4) as the updated hierarchical communities $\mathcal{H}_\ell(q)$ for q. Function QueryLabelRelated identifies the community C_ℓ. The function checks each query-labeled edge and if the edge is cut by an ancestor of q, the corresponding value in Δ is updated (Line 7-11). The *reclustering score* of each community $C \in \mathcal{H}(q)$ is then computed through dynamic programming (Line 12-18). The variable r memorizes the current reclustering score of each community and the community with largest r is returned as C_ℓ.

Algorithm 2: LORE

```
   Input: graph $g$, query $(q, \ell_q)$, community hierarchy $\mathcal{T}$
   Output: Label-aware hierarchical communities $\mathcal{H}_\ell(q)$
 1 $C_\ell \leftarrow$QueryLabeledRelated$(q, \ell_q, \mathcal{T})$;
 2 $S_\ell \leftarrow$ subgraph of $g_\ell$ induced by $C_\ell$;
 3 $\mathcal{T}_\ell \leftarrow$ HierarchicalClustering$(S_\ell)$;
 4 $\mathcal{H}_\ell(q) \leftarrow$ ancestors$(q, \mathcal{T}_\ell) \cup$ ancestors$(C_\ell, \mathcal{T})$;
 5 return $\mathcal{H}_\ell(q)$;

 6 Function QueryLabelRelated$(q, \ell_q, \mathcal{T})$:
 7     $\Delta \leftarrow$ an array of length $|\mathcal{H}(q)|$;
 8     foreach query-labeled edge $(u, v)$ do
 9         $d \leftarrow \mathbf{dep}(u, v)$;
10         if $q \in \boldsymbol{lca}(u, v)$ then
11             $\Delta[d] \leftarrow \Delta[d] + 1$;
12     $S \leftarrow 0, r \leftarrow 0$ ;
13     $C_\ell \leftarrow \varnothing$;
14     for $i$ in 0 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             $r \leftarrow \frac{S}{|C_i(q)|}$;
18             $C_\ell \leftarrow C_i(q)$;
19     return $C_\ell$;
```

Example 4.3. In Fig. 3, given the label ℓ_q ={DB} and the query node v_0, the algorithm first computes the reclustering score of communities containing v_0 in Fig. 1. We have $\Delta(C_3) = 1$ and $\Delta(C_4) = 2$. The reclustering scores $r(C_3) = \frac{\Delta(C_3)\cdot\mathbf{dep}(C_3)}{|C_3|} = \frac{3}{6} = \frac{1}{2}$. $r(C_4)$ is computed based on $r(C_3)$. We have $r(C_4) = \frac{r(C_3)\cdot|C_3|+\Delta(C_4)\cdot\mathbf{dep}(C_4)}{|C_4|} = \frac{3+2\cdot2}{8} = \frac{7}{8}$. Since $r(C_4) > r(C_3)$, C_4 is selected as the community C_ℓ. Then, the community C_ℓ is reclustered according to the weights in g_ℓ. Fig. 3 presents the result of the local reclustering. Community C'_0 to C'_3 are the communities obtained by reclustering C_4. Finally, we obtain hierarchical communities $\mathcal{H}_\ell(v_0) = \{C'_0, C'_3, C_4, C_6\}$ that reflect the query label information.

4.2 HIMOR-Index

Given the label-aware hierarchical communities $\mathcal{H}_\ell(q)$ obtained by LORE, we can directly perform Alg. 1 on $\mathcal{H}_\ell(q)$ to generate the characteristic community $C^*(q)$ for the query node. However, it could be time consuming for Alg. 1 to process $\mathcal{H}_\ell(q)$ containing very large communities.

Motivated by the observation that the label-aware $\mathcal{H}_\ell(q)$ returned by LORE is only influenced by the query label at deeper levels (descendants of C_ℓ) whereas the larger communities in $\mathcal{H}_\ell(q)$ (ancestors of C_ℓ) are unaffected since they come from the unlabeled community hierarchy, a straightforward indexing solution is to memorize the influence ranks of all nodes in each community from the unlabeled community hierarchy. For a query node q, we can first check if there exists an ancestor of C_ℓ such that q is top-k influential in this ancestor community. If such community exists, it is returned immediately as $C^*(q)$; otherwise, we execute Alg. 1 over the hierarchical communities in C_ℓ to compute $C^*(q)$. However, this straightforward indexing solution requires to memorize $O(|\mathcal{V}| \cdot h(\mathcal{T}))$ rank values, which takes large index spaces, where $h(\mathcal{T})$ denotes the height of $\mathcal{T}$.

HIMOR-index and query processing. In this section, we propose the *HIerarchical MOnotonic Ranking* (HIMOR) index, which accelerates the query processing for the COD problem over labeled graphs significantly. Instead of memorizing the all influence ranks of each node v, the HIMOR-index only memorizes the influence ranks at selected communities in $\mathcal{H}(v)$. In what follows, we present

v_0	v_1	v_2	v_3	v_4	v_5	v_6	v_7
C_6 2	C_6 5	C_6 5	C_6 2	C_6 7	C_6 7	C_6 1	C_6 2
C_0 1	C_3 3	C_0 3		C_1 1	C_1 1		C_3 1
	C_0 1						

Figure 4: The Monotonic community subset of nodes in Fig. 1.

the concept of monotonic community subset, which is the basis of HIMOR-index.

Definition 4.4 (Monotonic Community Subset). Given the unlabeled hierarchical communities $\mathcal{H}(q)$ of node q, the monotonic community subset $\mathcal{M}_q$ is subset of $\mathcal{H}(q)$ such that for any $C \in \mathcal{M}_q$ and $C' \in \mathcal{H}(q)$, if $C \subset C'$, we have $\mathbf{rank}(q, C) < \mathbf{rank}(q, C')$.

Example 4.5. In Fig. 1, given query node v_0 and the unlabeled hierarchical communities $\mathcal{H}(v_0) = \{C_0, C_3, C_4, C_6\}$, the influence rank of v_0 in each communities are $\mathbf{rank}(v_0, C_0) = 1$, $\mathbf{rank}(v_0, C_3) = 3$, $\mathbf{rank}(v_0, C4) = \mathbf{rank}(v_0, C6) = 2$ estimated with the RR graphs in Fig. 2(a). Thus, the monotonic community subset for node v_0 is $\mathcal{M}_{v_0} = \{(C_6, 2), (C_0, 1)\}$. $(C_3, 3)$ is not added into $\mathcal{M}_{v_0}$ since we have $\mathbf{rank}(v_0, C_6) < \mathbf{rank}(v_0, C_3)$ and $C_3 \subset C_6$. Fig. 4 presents the monotonic community subsets for each node in g in Fig. 1.

It is easy to see that $\mathcal{M}_q$ has the following property.

Property 1. *If the element (C, τ) for $\tau > k$ is contained in $\mathcal{M}_q$, then the ancestor communities of C in $\mathcal{H}(q)$ will not be the characteristic community for q since we have $\mathbf{rank}(q, C') > \mathbf{rank}(q, C) > k$ due to the monotonicity of $\mathcal{M}_q$.*

Thus, for each node $v \in \mathcal{V}_g$, we can select arbitrary subset of $\mathcal{M}_v$ as HIMOR-index. In the following, we use $\mathcal{M}_v$ and HIMOR-index at v interchangeably. For a query (q, ℓ_q), if we can find a item (τ, C) from $\mathcal{M}_q$ such that C is a super-community of C_ℓ and $\tau > k$, then we can perform Alg. 1 on $\mathcal{H}_\ell(q|C)$ to compute $C^*(q)$.

The pseudo code for COD query processing based on HIMOR-index is presented in Alg. 3. It takes graph g, a query (q, ℓ_q), parameter k and HIMOR-index $\mathcal{M}_q$ as input and outputs the characteristic community $C^*(q)$. The algorithm traverses $\mathcal{M}_q$ in descending order of the rank value (Line 1-8). If the current rank value is smaller than k, the current community C is the characteristic community (Line 2-4). Otherwise, if the next rank value r' is smaller than k or the next community C' is a subcommunity of C_ℓ, the compressed COD framework is performed over hierarchical communities in $\mathcal{H}_\ell(q)$ below community C and obtain $C^*(q)$ (Line 5-8).

Example 4.6. Given the query node v_0 and the parameter $k = 1$, the items in $\mathcal{M}_{v_0}$ is traversed from top to bottom. The algorithm first visit item $(C_6, 2)$, C_6 is not a characteristic community for v_0 since $k < 2$. The next item is $(C_0, 1)$ with community C_0 being a subcommunity of $C_\ell = C_3$ in $\mathcal{T}$. Thus, the compressed COD framework is performed over $\mathcal{H}_\ell(v_0|C_6)$ to obtain $C^*(v_0)$.

Compressed HIMOR-index Construction. A straightforward approach for HIMOR-index construction is to compute the monotonic community subset M_v for each node $v \in \mathcal{V}$ and then construct HIMOR-index at node v according to M_v. However, this approach requires influence estimation over $\mathcal{H}(v)$ for each node $v \in \mathcal{V}$, which is time prohibitive.

To this end, we propose a compressed construction algorithm for HIMOR. The key observation is that hierarchical communities for different nodes are highly overlapped with each other at shallow levels of the hierarchy. For two nodes $u, v \in \mathcal{V}$, hierarchical communities $\mathcal{H}(u)$ and $\mathcal{H}(v)$ will both contain ancestor communities of $\mathbf{lca}(u, v)$ in $\mathcal{T}$. The compressed HIMOR-index construction algorithm have two major steps similar to the compressed COD framework for query processing but with two crucial differences:

Algorithm 3: COD based on HIMOR

```
  Input: graph $g$, query $(q, \ell_q)$, parameter $k$ and monotonic
         community subset $\mathcal{M}_q$
  Output: characteristic community $C^*(q)$
1 for $(\tau, C) \in \mathcal{M}_q$ do
2     if $\tau \le k$ then
3         $C^*(q) \leftarrow C$;
4         break;
5     let $(\tau', C')$ be the next element in $\mathcal{M}_q$;
6     if $C' \subsetneq C_\ell$ or $r' < k$ then
7         $C^*(q) \leftarrow$ perform Alg. 1 on $\mathcal{H}_\ell(q|C)$;
8         break;
9 return $C^*(q)$;
```

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

We describe the compressed construction algorithm as follows. *First*, it generates Θ RR graphs and performs 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 small to large. To enable $O(1)$ insertion time for the queue, we maintain $|\mathcal{T}|$ hash maps similar to Alg. 1. When node v tagged with C_i in the queue is explored, HFS records the influence of v in bucket B_i, visits each $u \in \mathcal{N}(v)$, and adds $(u, \mathbf{lca}(u, C_i))$ into 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 as a tree corresponding to each community in $\mathcal{T}$. *Second*, the algorithm processes buckets produced by HFS in a bottom-up manner. The processing of each bucket B_i contains three steps: (1) update $B_i(v)$ to $B_i(v) + B_j(v)$ for node v appears in both bucket B_i and its child 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. Thereafter, $\mathcal{M}_v$ for each node v is computed based on its influence ranks.

Example 4.7. The tree-structured buckets generated through HFS is presented in the left subfigure in Fig. 5. Bucket B_4 contains items $(v_2, 1)$, $(v_3, 1)$, $(v_0, 1)$ and $(v_6, 1)$ recorded during the processing of HFS on RR graph (2) in Fig. 2(a). Note that items $(v_5, 1)$ and $(v_4, 1)$ are recorded in bucket B_1 and are incrementally added into B_4 in the second stage. Initially, buckets B_0 and B_2 are merged into bucket B_3. Since node v_7 appears in both B_3 and B_2, the algorithm updates $B_3(v_7) = 3$. Similarly, the algorithm updates $B_3(v_6) = 3$ and $B_3(v_3) = 2$. The up-to-date bucket B_3 is then sorted to $B_3 = \{(v_7, 3), (v_6, 3), (v_3, 2)\}$. Then, buckets B_0 and B_2 are combined with B_3 by *merge sort* to obtain the influence ranks of nodes in C_3. The middle subfigure in Fig. 5 shows the buckets after the processing

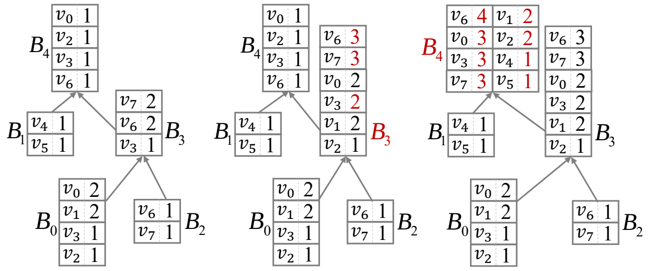


Figure 5: The tree-structured buckets.

Table 2: Network statistics.

Network	$\lvert\mathcal{V}\rvert$	$\lvert\mathcal{E}\rvert$	$\lvert\mathcal{L}\rvert$	$\lvert\bar{\mathcal{H}}_\ell(q)\rvert$
`cora`	2485	5069	7	18.524
`citeseer`	2110	3668	6	18.89
`pubmed`	19717	44327	3	34.225
`retweet`	18470	48053	2	165.3
`amazon`	334863	925872	33	54.8
`dblp`	317080	1049866	31	47.9

of B_3. Bucket B_4 is processed similarly and the result are displayed in the right subfigure of Fig. 5.

Time Complexity. The first stage for compressed HIMOR-index construction requires $O(\Theta \cdot \omega + |\mathcal{R}|)$ operations, which is the same to the influence estimation stage of compressed COD framework for query processing. Note that the HFS algorithm requires the **lca**-operation to compute the community tag for each node added into the queue. Following the indexing method in [3], each **lca**-operation executes in $O(1)$ time, and thus does not affect the time complexity of the first stage. In the second stage, to process bucket B_i, the algorithm first updates the values in B_i and then sorts the updated B_i, which requires $O(|B_i| + |B_i| \cdot \log |B_i|) = O(|B_i| \cdot \log |B_i|)$ operations. Then, the algorithm merges B_i with its child buckets. During the second stage, node v is merged $\mathbf{dep}(v)$ times in total. Thus, the total time complexity of the second stage is $O(\sum_{B_i} |B_i| \cdot \log |B_i| + \sum_{v \in \mathcal{V}} \mathbf{dep}(v)) = O(|\mathcal{R}| \cdot \log |\mathcal{V}| + \sum_{v \in \mathcal{V}} \mathbf{dep}(v))$. The total time complexity of compressed HIMOR-index construction is $O(\Theta \cdot \omega + |\mathcal{R}| \cdot \log |\mathcal{V}| + \sum_{v \in \mathcal{V}} \mathbf{dep}(v))$.

Space Complexity. The size of the HIMOR-index can be tuned easily by users since any subset of $\mathcal{M}_q$ can be used as HIMOR-index for query node q. Given a hyperparameter T, we choose T layers evenly from M_v as the HIMOR-index for each node v. Thus, the space complexity is $O(T \cdot |\mathcal{V}|)$.

5 EXPERIMENTS

For our experimental evaluations, we focus on the COD problem over labeled graphs. The COD problem over unlabeled graphs can be regarded as a special case of the labeled problem by assigning each node in the unlabeled graph with the same label.

To evaluate the effectiveness and efficiency of our proposed hierarchical approaches for processing COD, we conduct extensive experiments on real-world datasets with both real and synthetically generated node labels. Sec. 5.1 describes the experimental setup. The remaining parts of the section are presented to answer the following research questions:

- Can the proposed hierarchical approaches find better characteristic communities than the existing methods on community search? (Sec. 5.2)?
- How effective is the compressed COD framework against the naïve independent evaluation approach (Sec. 5.3)?
- Can the optimizations, i.e., local reclustering and index, significantly accelerate the query processing for COD (Sec. 5.4)?
- Does the hyperparameter λ for label-aware hierarchical clustering influence the performance of our hierarchical methods significantly? (Sec. 5.5)
- Is there a proper value of T that can strike a balance between the effectiveness and size of HIMOR-index (Sec. 5.6)?

5.1 Experimental Setup

Datasets. We conduct our experiments with six real-world network datasets. The network statistics are reported in Table 2. Note that $\mathcal{L}$ denotes the label set for each dataset and $|\bar{\mathcal{H}}_\ell(q)|$ refers to the average number of hierarchical communities containing a query node. Details of the datasets are presented as follows:

The first four datasets are networks with real-world node labels: `cora`, `citeseer`, `pubmed` and `soc-political-retweet` (abbreviated as `retweet`). The four datasets are available from the Network Repository [2]. The remaining two datasets `amazon` and `dblp` are real-world networks that contain ground-truth communities but do not have real-world node labels. To this end, we follow the setup used in [20, 41] to augment node labels for these datasets. We generate a set of labels consisting of $|\mathcal{L}| = 0.0001 \cdot |\mathcal{V}|$ distinct labels for each network. For each ground-truth community in the network, we randomly select a label from $\mathcal{L}$ and assign this label to nodes in the ground-truth community. These two datasets can be downloaded from the Stanford Network Analysis Project [3].

Query Generation. For each dataset, we randomly select 100 nodes from the network as query nodes. For each query node q, we then randomly pick a node label from q's labels as the query label.

Hierarchical Communities Generation. For each query with label ℓ_q, we first transform a labeled graph g to a weighted graph g_ℓ. We adopt an intuitive transformation to fuse node labels into edge weights of g_ℓ. Given a real number $\lambda \in (0, 1)$, $g_\ell = (\mathcal{V}_{g_\ell}, \mathcal{E}_{g_\ell})$ is a weighted graph with $\mathcal{V}_{g_\ell} = \mathcal{V}_g$ and $\mathcal{E}_{g_\ell} = \mathcal{E}_g$. For each $(u, v) \in \mathcal{E}_{g_\ell}$, $w(u, v) = 1$ if both u and v have label ℓ_q; otherwise, $w(u, v) = 1 - \lambda$. Note that our algorithm is orthogonal to the transformation scheme and thus can also be combined with more sophisticated schemes [5, 41].

We then obtain the hierarchical communities $\mathcal{H}_\ell(q)$ by processing g_ℓ with hierarchical agglomerative clustering In particular, we use the linkage function from [24] and the nearest neighbor chain algorithm [16, 29]. The choice of the hierarchical clustering algorithm is also orthogonal to our contributions.

Compared Methods. To the best of our knowledge, this is the first work on the COD problem and there is no direct comparable methods. Thus, we compare our proposed hierarchical approaches with the existing methods of community search on labeled graphs,

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

[3] http://snap.stanford.edu/

Table 3: Parameters used in experiments.

Param.	Description	Default
k	influence rank of q in $C^*(q)$	5
θ	expected number of $\mathcal{R}_g$ generated from each node, i.e., $\Theta = \theta \cdot \vert\mathcal{V}\vert$	10
T	number of items in HIMOR-index for each node	10
λ	parameter for labeled hierarchical clustering	0.9

which aims at finding a densely connected subgraph that containing the query node and with high label density.

- ACQ [12] produces a k-core community containing the query node such that all nodes in the returned community share the query label.
- ATC [20] returns a (k, d)-truss community containing the query node and maximizes a dedicated label score function.
- CAC [42] finds a triangle-connected k-truss containing the query node such that all nodes in the returned community share the query label.

We also compare different variants of the hierarchical approaches for the COD problem proposed in this work.

- CODU generates the unlabeled hierarchical communities $\mathcal{H}(q)$ by hierarchical clustering of the original graph g and then obtains $C^*(q)$ from $\mathcal{H}(q)$.
- CODR generates label-aware hierarchical communities $\mathcal{H}_\ell(q)$ by hierarchical clustering of the transformed graph g_ℓ from scratch and obtains $C^*(q)$ from $\mathcal{H}_\ell(q)$.
- CODL generates $\mathcal{H}_\ell(q)$ through the LORE algorithm and obtains $C^*(q)$ from $\mathcal{H}_\ell(q)$ utilizing the HIMOR-index.

Both CODU and CODR also use the compressed COD framework as CODL to obtain $C^*(q)$ by default since the naïve independent evaluation approach is time-prohibitive.

Parameter Setup. We set $k = 5$ by default as the influence rank in COD queries. Parameter θ, which denotes the expected number of RR graphs generated from each node for influence estimation, is set to 10 by default. We follow previous works [8, 14, 38] and use the weighted cascade model to set influence probabilities in the IC model. Specifically, for each edge $(u, v) \in \mathcal{E}$, the influence probability $p(u, v)$ is set to $\frac{1}{|\mathcal{N}(v)|}$. Parameter T that controls the size of the HIMOR-index is set to 10 to strike a balance between the effectiveness and size of the HIMOR-index as discussed in Sec. 5.6. We set $\lambda = 0.9$ for labeled hierarchical clustering. The parameters and their default values are summarized in Table 3.

Evaluation Metrics. We use three metrics over the result characteristic community $C^*(q)$ to quantify the effectiveness of different compared methods:

- The size of the result characteristic community $|C^*|$, i.e., the number of nodes in $C^*(q)$;
- The topology density $\rho(C^*)$, which is the ratio between number of edges and number of nodes in $C^*(q)$;
- The label density $\varphi(C^*)$, i.e., the number of query labels in $C^*(q)$ divided by the number of nodes in $C^*(q)$.

In the experiments, we report the average of each metric over 100 queries for each compared method. In the case where the community search methods cannot find the characteristic community for a query, we record the values of the evaluation metrics as 0.

Environment.. All experiments are conducted on a Linux Server with AMD EPYC 7643 CPU and 256 GB memory running Ubuntu 20.04. We use the original implementation of CAC [42], ATC [20] (implemented with C++) and ACQ [12] (implemented with Java) provided by the authors. Other algorithms are implemented with Python and executed with a single thread.

5.2 Overall Effectiveness Evaluation

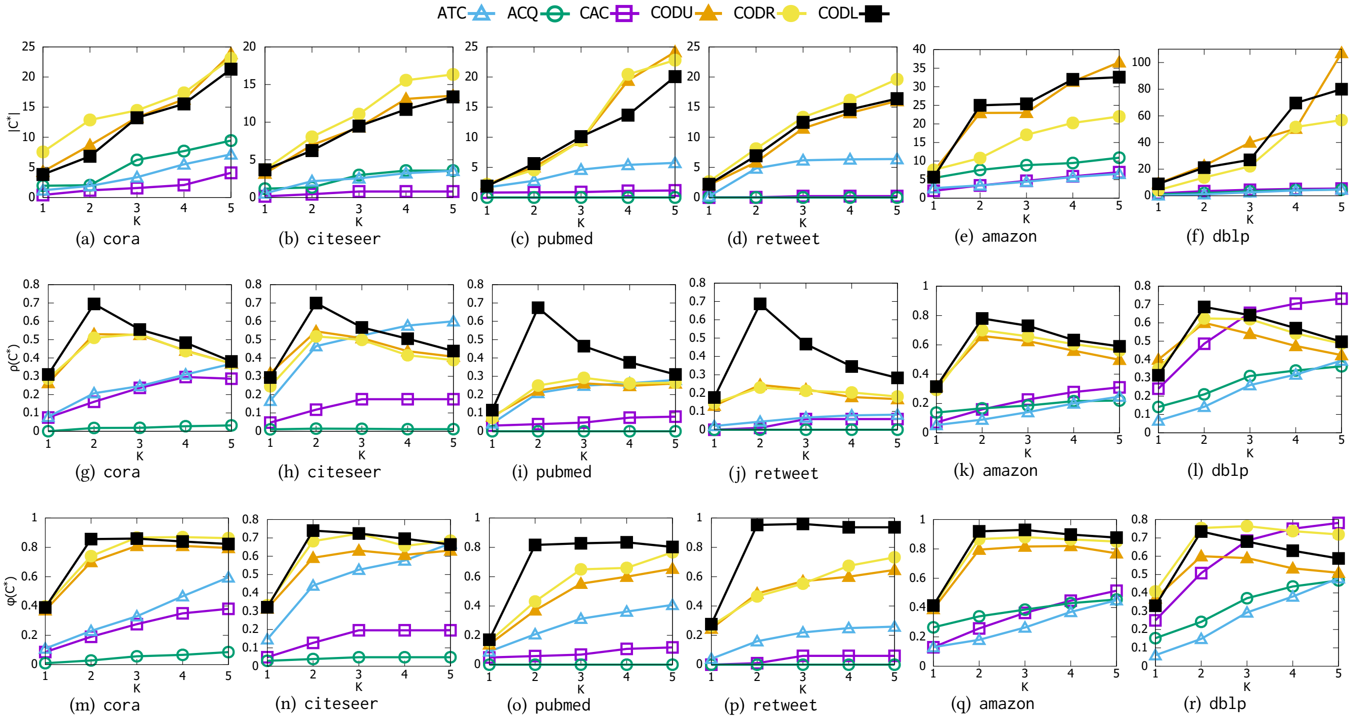
We evaluate the overall effectiveness of our proposed method CODL against both labeled community search methods (CAC, ACQ and ATC) and variants of the hierarchical approaches (CODU and CODR). The purpose is to demonstrate: (1) the effectiveness of the hierarchical approaches for the COD problem (through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

Average size of $|C^*|$. Fig. 6(a)-Fig. 6(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 while searching for the result community and thus cannot find the characteristic community for many queries. We can also observe that the average size of $C^*(q)$ obtained by different methods increase monotonically with parameter k. With a less restricted requirement on the influence rank of the query node q, larger characteristic communities can be discovered.

Average topology density $\rho(C^*)$. Fig. 6(g)-Fig. 6(l) display the average topology density $\rho(C^*)$ for different methods across datasets and the result shows that our proposed method CODL achieves the largest $\rho(C^*)$ under most circumstances.

First, CODL performs better than CODU and CODR across all datasets, which verifies the effectiveness of the LORE algorithm. Specifically, CODL achieves the peak value of topology density at $k = 2$ and outperforms baselines 71.62% on average across datasets. CODR returns characteristic communities with unsatisfactory $\rho(C^*)$ since it reclusters the transformed graph g_ℓ from scratch and thus results in the free-rider effect, i.e., adding nodes that have low structure relevance with the query node into communities at deeper levels of $\mathcal{H}_\ell(q)$. CODU does not consider the label information at all in $\mathcal{H}(q)$. Failure to utilize the label information results in finding community with sub-optimal quality, which agrees with the findings from previous works on community analysis [12, 20]. In contrast, CODL takes useful label information into consideration and avoids the free-rider effect by locally reclustering the transformed graph g_ℓ.

Second, CODL achieves larger $\rho(C^*)$ than the community search methods CAC, ACQ and ATC over 4 out of 6 datasets. For small k, many communities found by the community search methods are not characteristic communities wrt. k and thus lower density values are displayed. When k becomes larger, ATC and CAC achieve larger $\rho(C^*)$ than CODL on `citeseer` and `dblp` respectively. The reason is that ATC and CAC adopt the strict community models of (k, d)-truss and triangle connected truss for community search and thus will return small and dense communities. With less restriction on the influence rank wrt. k, the baselines can find communities that are eligible characteristic communities and produce high values of



(a) cora (b) citeseer (c) pubmed (d) retweet (e) amazon (f) dblp

(g) cora (h) citeseer (i) pubmed (j) retweet (k) amazon (l) dblp

(m) cora (n) citeseer (o) pubmed (p) retweet (q) amazon (r) dblp

Figure 6: Overall effectiveness of our hierarchical methods compared with community search methods varying k. Subfigures (a)-(f) present the average size $|C^*|$. Subfigures (g)-(l) present the average topology density $\rho(C^*)$. Subfigures (m)-(r) present the average label density $\varphi(C^*)$.

$\rho(C^*)$. Nevertheless, $C^*(q)$ obtained by ATC and CAC on citeseer and dblp are much smaller in terms of size than $C^*(q)$ obtained by CODL according to Fig. 6(b) and Fig. 6(f).

Average label density $\varphi(C^*)$**.** Fig. 6(m)-Fig. 6(r) display the average label density $\varphi(C^*)$ of different methods on each dataset and the result again confirms the superiority of CODL. CODL achieves better or comparable values of $\varphi(C^*)$ against the baselines across all datasets except on dblp. CODU achieves sub-optimal $\varphi(C^*)$ since it does not consider the query labels at all while generating $\mathcal{H}(q)$. We want to highlight that it is impressive for CODL to outperform CODR in terms of $\varphi(C^*)$ over most datasets.

Although CODR places more emphasis on the query labels by reclustering the transformed graph g_ℓ from scratch, it can add free-rider nodes without query labels but are close to nodes with query labels into deeper communities in $\mathcal{H}_\ell(q)$. On datasets citeseer and dblp, the community search methods ATC and CAC again achieve comparable or higher $\varphi(C^*)$ than CODL due to the reason discussed in the part **Average topology density** $\rho(C^*)$.

5.3 Compressed COD Framework Evaluation

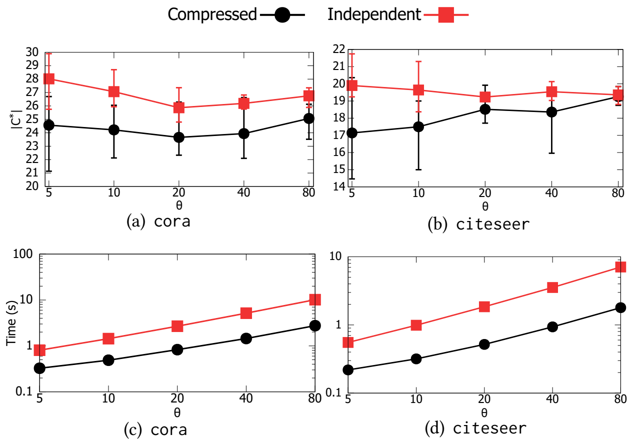
In this section, we evaluate the effectiveness of the compressed COD framework proposed in Sec. 3 by comparing the followings:

- Compressed is a variant of CODR which uses the compressed COD framework for $C^*(q)$ discovery. Compressed generates $\Theta = \theta \cdot |\mathcal{V}|$ samples to estimate the influence on all communities from $\mathcal{H}_\ell(q)$.
- Independent is a variant of CODR by independently evaluating the influence rank of the query node in each community from $\mathcal{H}_\ell(q)$. In total, Independent generates $\Theta = \theta \cdot \sum_{C \in \mathcal{H}_\ell(q)} |C|$ samples for influence estimation.

The aim is to show that our compressed COD framework can reduce the number of sampled RR sets significantly while preserving the effectiveness of the influence estimation. We focus on the smaller datasets cora, citeseer for the evaluation since the Independent approach is not scalable to large graphs.

Fig. 7(a)-7(b) presents the average size of $C^*(q)$ discovered by the Compressed and Independent approaches when varying the number of samples generated for influence estimation. We can observe that the Independent approach finds relatively larger $C^*(q)$ than the Compressed approach. Given the same value of θ, the Independent approach repeats the process for $|\mathcal{H}_\ell(q)|$ communities whereas the Compressed reuses the samples. Thus, the Independent approach generates much more samples than the Compressed approach for influence estimation and achieves relatively better performance. Nevertheless, with the increase of the sample size, the Compressed approach converges to the Independent approach and the gap between the two approaches is small.

Fig. 7(c)-7(d) presents the execution time of the Compressed and Independent approaches when varying the number of samples. We observe that: (1) Compressed is significantly more efficient than Independent; (2) the speedup ratio of Compressed over Independent remains constant for more samples. Specifically, when $\theta = 80$, Compressed takes 2.76 and 1.79 seconds for query processing on cora



(a) cora (b) citeseer

(c) cora (d) citeseer

Figure 7: Comparisons between the Compressed and Independent approaches when varying θ. (a)-(b) show average sizes of returned $C^*(q)$ and error bars denote the maximum and minimum size; (c)-(d) compare the execution time.

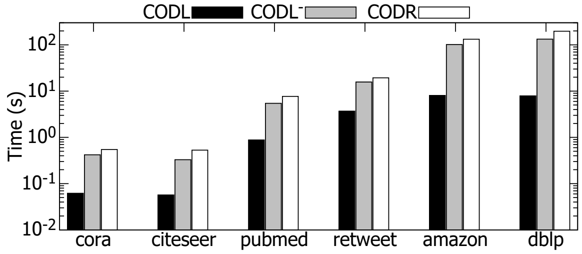


Figure 8: The execution time of different local hierarchical reclustering optimizations.

and citeseer respectively; whereas Independent requires 10.14 and 7.08 seconds respectively. Thus, Compressed is more than 3 times faster than Independent over cora and citeseer. For larger datasets, Compressed achieves higher speedups. For example, Compressed is faster than Independent by 10 and 9 times on pubmed and retweet, respectively. Furthermore, Independent cannot terminate within 36 hours limit for the dataset amazon and dblp with $\theta = 10$. 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 takes 132.03 and 196.34 seconds for Compressed to process a query on average on amazon and dblp respectively. In Sec. 5.4, we show that the execution time can be further reduced with the optimizations.

5.4 Optimization Evaluations

In this section, we verify the efficiency improvement through utilizing local hierarchical reclustering optimizations proposed in Sec. 4. We compare our fully optimized algorithm CODL with baselines CODL$^-$ and CODR, where CODL$^-$ obtains the hierarchical communities $\mathcal{H}_\ell(q)$ by the LORE algorithm and use the compressed COD framework to discover $C^*(q)$ without HIMOR-index.

Efficiency Study. Fig. 8 displays the execution time of each method across different datasets. It is apparent that the fully optimized algorithm CODL is significantly more efficient than the baselines. Specifically, on dblp, CODL takes 7.9 seconds on average for query processing, whereas CODR requires 196.34 seconds. Thus, fully optimized CODL is 25 times faster than CODR. We then discuss each optimization in detail.

CODL is faster than CODL$^-$ by at least 5 times and up to 10 times on larger datasets (amazon and dblp). The reason is that by utilizing the HIMOR-index, CODL only has to perform the compressed COD framework over the small community identified from $\mathcal{M}_q$ whereas CODL$^-$ has to perform the compressed COD framework over all hierarchical communities from $\mathcal{H}_\ell(q)$. CODL$^-$ is faster than CODR since CODR has to perform hierarchical clustering over g_ℓ from scratch, whereas CODL$^-$ only needs to recluster the community C_ℓ while generating the hierarchical community set $\mathcal{H}_\ell(q)$. The result reveals that LORE not only finds better characteristic community but also improves the efficiency.

Table 4: Time and Memory Overhead of HIMOR-Index.

Network	Time (s)	Mem (MB)
cora	0.81	0.15
citeseer	0.53	0.12
pubmed	16.18	1.29
retweet	113.42	1.31
amazon	317.63	24.55
dblp	441.23	22.40

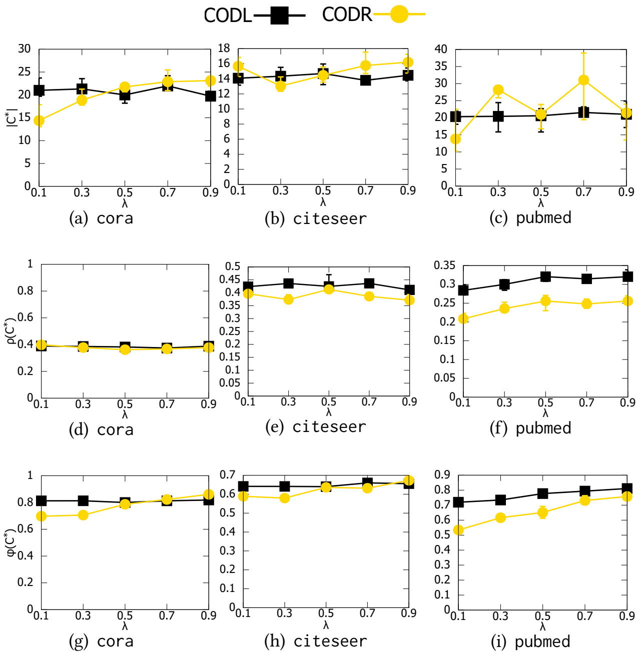
Index Overhead. Table 4 presents the index construction time and the memory cost for utilizing the index for query processing on all datasets. The result shows that HIMOR-index can be constructed efficiently and only lead to reasonable memory overhead for query processing. Specifically, on datasets amazon and dblp with over 300k nodes, it takes less than 10 minutes for the index construction and the memory overhead is less than 25MB.

In addition, we observe that HIMOR-index construction on retweet is more time consuming than that on pubmed with similar size. According to Sec. 4.2, the time complexity of index construction is related to $\sum_{v \in \mathcal{V}_g} \mathbf{dep}(v)$, which reflects the degree of balance of $\mathcal{T}$. In Table 2, we can observe that the average depth $|\mathcal{H}_\ell(q)|$ for 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 lead to a relatively larger time consumption. Finding balanced hierarchical communities is orthogonal to our work and is studied in literature [22]. Nevertheless, we can potentially integrate balanced hierarchical clustering to accelerate the index construction process.

5.5 Parameter Tuning for λ

We vary the hyperparameter used in the label-aware hierarchical clustering and report the performance of the CODL and CODR over datasets cora, citeseer and pubmed in Fig. 9.

We have the following observations. *First*, the performance of CODL is less affected by the hyperparameter λ than the performance of CODR. The reason is that CODR performs the label-aware hierarchical clustering over the whole transformed graph g_ℓ and thus is more influenced by the hyperparameter λ; whereas CODL only recluster the small community C_ℓ and thus is less affected



(a) cora (b) citeseer (c) pubmed

(d) cora (e) citeseer (f) pubmed

(g) cora (h) citeseer (i) pubmed

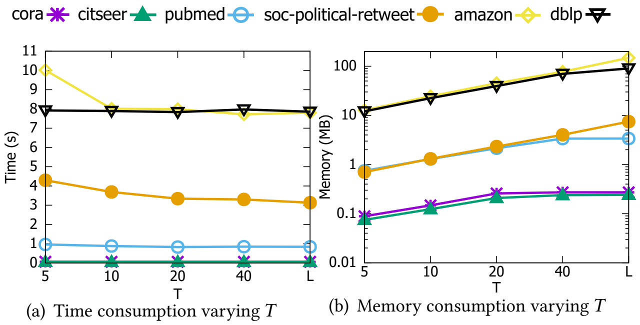
Figure 9: Performance of CODL and CODR varying λ. Subfigures (a)-(c) present the average size $|C^*|$, subfigures (d)-(f) present the average topology density $\rho(C^*)$ and subfigures (g)-(i) present the average label density $\varphi(C^*)$.

by λ. *Second*, the performance of CODL and CODR are positively correlated with λ under most circumstances. The reason is that with a larger value of λ, the label-aware hierarchical clustering can utilize the label information to generate communities with better quality. Thus, we set $\lambda = 0.9$ by default during other experiments.

5.6 Parameter Tuning for T in HIMOR- Index

We vary the hyperparameter T used in the construction of HIMOR-index and report the effectiveness of the HIMOR-index in Fig. 10, which is measured by the time consumption of CODL algorithm and the size of the HIMOR-index.

We have the following observations. On the one hand, CODL generally becomes more time efficient with the increase of T. The reason is that with a larger T, CODL can identify a smaller community from $\mathcal{H}_\ell(q)$ as the input for the compressed COD framework based on the HIMOR-index. Nevertheless, we can observe that the effectiveness of HIMOR-index does not increase linearly to T. On datasets except amazon, the time consumption of CODL is relatively stable when T is varied from 5 to L. On amazon, the time consumption of CODL goes through a large drop when T is varied from 5 to 10, and keeps stable when T is further increased. On the other hand, the size of the HIMOR-index grows with the increase of T since T represents the upper bound of the size of HIMOR-index at each node. Based on the two observations, we strike a balance between the size of the HIMOR-index and time efficiency of CODL algorithm and set $T = 10$ by default.



(a) Time consumption varying T (b) Memory consumption varying T

Figure 10: Time and memory consumption varying T. The value L of parameter T denotes that the monotonic community subset $\mathcal{M}_v$ is used as the HIMOR-index for each node v directly.

6 CONCLUSION AND FUTURE WORK

We introduced and studied the novel problem of characteristic community discovery (COD), which seeks the largest community within a hierarchy, in which a node is impactful. We proposed the compressed COD framework that solves COD efficiently over hierarchical communities. To address COD over labeled graphs, we crafted the LORE algorithm, which eschews the free-rider effect while considering node labels. Lastly, we accelerate COD query processing over labeled graphs by the proposed HIMORindex. Extensive experiments on real-world data verified the effectiveness and efficiency of our solutions.

Future work. We focus on discovering the characteristic community based on the influence diffusion models such as the IC model. Other models have been proposed to measure the importance of nodes such as closeness centrality [32], betweenness centrality [34] and PageRank [33]. In the future, we aim to address COD under those models.

REFERENCES

[1] Yong-Yeol Ahn, James P Bagrow, and Sune Lehmann. 2010. Link communities reveal multiscale complexity in networks. *nature* 466, 7307 (2010), 761–764.

[2] Vladimir Batagelj and Matjaz Zaversnik. 2003. An O (m) algorithm for cores decomposition of networks. *arXiv preprint cs/0310049* (2003).

[3] Michael A Bender, Martin Farach-Colton, Giridhar Pemmasani, Steven Skiena, and Pavel Sumazin. 2005. Lowest common ancestors in trees and directed acyclic graphs. *Journal of Algorithms* 57, 2 (2005), 75–94.

[4] Christian Borgs, Michael Brautbar, Jennifer Chayes, and Brendan Lucier. 2014. Maximizing social influence in nearly optimal time. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*. SIAM, 946–957.

[5] Cécile Bothorel, Juan David Cruz, Matteo Magnani, and Barbora Micenkova. 2015. Clustering attributed graphs: models, measures and methods. *Network Science* 3, 3 (2015), 408–444.

[6] Lijun Chang, Xuemin Lin, Lu Qin, Jeffrey Xu Yu, and Wenjie Zhang. 2015. Index-based optimal algorithms for computing steiner components with maximum connectivity. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. 459–474.

[7] Wei Chen, Wei Lu, and Ning Zhang. 2012. Time-critical influence maximization in social networks with time-delayed diffusion process. In *Twenty-Sixth AAAI Conference on Artificial Intelligence*.

[8] Wei Chen, Chi Wang, and Yajun Wang. 2010. 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*. 1029–1038.

[9] Aaron Clauset. 2005. Finding local community structure in networks. *Physical review E* 72, 2 (2005), 026132.

[10] Jonathan Cohen. 2008. Trusses: Cohesive subgraphs for social network analysis. *National security agency technical report* 16, 3.1 (2008).

[11] Laxman Dhulipala, David Eisenstat, Jakub Łącki, Vahab Mirrokni, and Jessica Shi. 2021. Hierarchical agglomerative graph clustering in nearly-linear time. In *International Conference on Machine Learning*. PMLR, 2676–2686.

[12] Yixiang Fang, Reynold Cheng, Siqiang Luo, and Jiafeng Hu. 2016. Effective community search for large attributed graphs. *Proceedings of the VLDB Endowment* 9, 12 (2016), 1233–1244.

[13] Jacob Goldenberg, Barak Libai, and Eitan Muller. 2001. Talk of the network: A complex systems look at the underlying process of word-of-mouth. *Marketing letters* 12, 3 (2001), 211–223.

[14] Amit Goyal, Francesco Bonchi, and Laks VS Lakshmanan. 2011. A data-based approach to social influence maximization. *arXiv preprint arXiv:1109.6886* (2011).

[15] Mark Granovetter. 1978. Threshold models of collective behavior. *American journal of sociology* 83, 6 (1978), 1420–1443.

[16] Ilan Gronau and Shlomo Moran. 2007. Optimal implementations of UPGMA and other common clustering algorithms. *Inform. Process. Lett.* 104, 6 (2007), 205–210.

[17] Jiafeng Hu, Xiaowei Wu, Reynold Cheng, Siqiang Luo, and Yixiang Fang. 2016. Querying minimal steiner maximum-connected subgraphs in large graphs. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*. 1241–1250.

[18] Keke Huang, Sibo Wang, Glenn Bevilacqua, Xiaokui Xiao, and Laks VS Lakshmanan. 2017. Revisiting the stop-and-stare algorithms for influence maximization. *Proceedings of the VLDB Endowment* 10, 9 (2017), 913–924.

[19] Xin Huang, Hong Cheng, Lu Qin, Wentao Tian, and Jeffrey Xu Yu. 2014. Querying k-truss community in large and dynamic graphs. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 1311–1322.

[20] Xin Huang and Laks VS Lakshmanan. 2017. Attribute-driven community search. *Proceedings of the VLDB Endowment* 10, 9 (2017), 949–960.

[21] David Kempe, Jon Kleinberg, and Éva Tardos. 2003. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. 137–146.

[22] Marina Knittel, John P Dickerson, and MohammadTaghi Hajiaghayi. 2022. Generalized Reductions: Making any Hierarchical Clustering Fair and Balanced with Low Cost. *arXiv preprint arXiv:2205.14198* (2022).

[23] Renaud Lambiotte. 2010. Multi-scale modularity in complex networks. In *8th International symposium on modeling and optimization in mobile, ad hoc, and wireless networks*. IEEE, 546–553.

[24] Bastian Leibe, Krystian Mikolajczyk, and Bernt Schiele. 2006. Efficient clustering and matching for object class recognition.. In *BMVC*. 789–798.

[25] Yanhua Li, Wei Chen, Yajun Wang, and Zhi-Li Zhang. 2013. 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*. 657–666.

[26] Yuchen Li, Ju Fan, Yanhao Wang, and Kian-Lee Tan. 2018. Influence maximization on social graphs: A survey. *IEEE Transactions on Knowledge and Data Engineering* 30, 10 (2018), 1852–1872.

[27] Wei Lu, Francesco Bonchi, Amit Goyal, and Laks VS Lakshmanan. 2013. 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*. 928–936.

[28] Feng Luo, James Z Wang, and Eric Promislow. 2008. Exploring local community structures in large networks. *Web Intelligence and Agent Systems: An International Journal* 6, 4 (2008), 387–400.

[29] Fionn Murtagh. 1983. A survey of recent advances in hierarchical clustering algorithms. *The computer journal* 26, 4 (1983), 354–359.

[30] Mark EJ Newman and Michelle Girvan. 2004. Finding and evaluating community structure in networks. *Physical review E* 69, 2 (2004), 026113.

[31] Hung T Nguyen, My T Thai, and Thang N Dinh. 2016. Stop-and-stare: Optimal sampling algorithms for viral marketing in billion-scale networks. In *Proceedings of the 2016 international conference on management of data*. 695–710.

[32] Paul W Olsen, Alan G Labouseur, and Jeong-Hyon Hwang. 2014. Efficient top-k closeness centrality search. In *2014 IEEE 30th International Conference on Data Engineering*. IEEE, 196–207.

[33] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank citation ranking: Bringing order to the web.* Technical Report. Stanford InfoLab.

[34] Matteo Riondato and Evgenios M Kornaropoulos. 2014. Fast approximation of betweenness centrality through sampling. In *Proceedings of the 7th ACM international conference on Web search and data mining*. 413–422.

[35] Stephen B Seidman. 1983. Network structure and minimum degree. *Social networks* 5, 3 (1983), 269–287.

[36] Herbert A Simon. 1991. The architecture of complexity. In *Facets of systems science*. Springer, 457–476.

[37] Mauro Sozio and Aristides Gionis. 2010. 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*. 939–948.

[38] Youze Tang, Yanchen Shi, and Xiaokui Xiao. 2015. Influence maximization in near-linear time: A martingale approach. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*. 1539–1554.

[39] You Wu, Pankaj K Agarwal, Chengkai Li, Jun Yang, and Cong Yu. 2012. On" one of the few" objects. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1487–1495.

[40] Yubao Wu, Ruoming Jin, Jing Li, and Xiang Zhang. 2015. Robust local community detection: on free rider effect and its elimination. *Proceedings of the VLDB Endowment* 8, 7 (2015), 798–809.

[41] Niu Yudong, Yuchen Li, Ju Fan, and Zhifeng Bao. 2022. Local Clustering over Labeled Graphs: An Index-Free Approach. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 2805–2817.

[42] Yuanyuan Zhu, Jian He, Junhao Ye, Lu Qin, Xin Huang, and Jeffrey Xu Yu. 2020. When structure meets keywords: Cohesive attributed community search. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 1913–1922.