

# HOCTracker: Tracking the Evolution of Hierarchical and Overlapping Communities in Dynamic Social Networks

Sajid Yousuf Bhat and Muhammad Abulaish, *Senior Member, IEEE*

**Abstract**—In this paper, we propose a unified framework, HOCTracker, for tracking the evolution of hierarchical and overlapping communities in online social networks. Unlike most of the dynamic community detection methods, HOCTracker adapts a preliminary community structure towards dynamic changes in social networks using a novel density-based approach for detecting overlapping community structures, and automatically tracks evolutionary events like *birth*, *growth*, *contraction*, *merge*, *split*, and *death* of communities. It uses a novel and efficient log-based approach to map evolutionary relations between communities identified at two consecutive time-steps of a dynamic network, which considerably reduces the number of community comparisons. Moreover, it does not require an ageing function to remove old interactions for identifying community evolutionary events. HOCTracker is applicable to directed/undirected and weighted/unweighted networks. Experimental results have shown that community structures identified by HOCTracker on some well-known benchmark networks are significant and in general better than the community structures identified by the state-of-the-art methods.

**Index Terms**—Social network analysis, Overlapping community detection, Community hierarchy, Community evolution

## 1 INTRODUCTION

COMMUNITY mining research from social networks has recently gained significant popularity with an aim of analyzing mesoscopic structure of networks and their evolution. As a result, numerous methods related to community mining have been proposed in literature [1], [2]. Communities in social networks often map to important functional groups making their detection a highly desirable task. However, the problem of community detection in social networks depends on various factors, including whether the definition of community relies on global or local network properties, whether communities overlap, whether communities possess a hierarchical structure, whether link weights are utilized, whether outliers are considered, and whether dynamic nature of networks/communities is considered.

### 1.1 Overlapping and Hierarchical Communities

An open challenge related to community detection is the identification of overlapping communities which exist when a particular node of a network simultaneously belongs to several communities. The most popular method for identifying overlapping communities is the CPM (aka CFinder), which is based on percolating  $k$ -cliques (i.e., a complete subgraph of  $k$  nodes) from an underlying network [3]. Later on,

various methods including MOSES [4] and SHRINK [5] were proposed to identify overlapping communities in social networks. Recently, Xie et al. [6] and Gregory [7] proposed overlapping community detection methods SLPA and COPRA, respectively that are based on label propagation, wherein community labels are propagated between nodes according to pairwise interaction rules.

None of the methods mentioned above consider the hierarchical structure of communities. However, in order to provide appropriate information about the modular structure of a network, it is desirable to detect overlapping communities along with their hierarchical organization wherein multiple smaller communities are embedded within larger communities or a community may be a part of even larger communities. Kumar et al. [8] proposed HOC which uses a *topological overlap* criteria to define the similarity between two arbitrary nodes in a network to identify clique-based communities. Lancichinetti et al. [9] presented OSLOM which is able to detect a hierarchical community structure by repeatedly applying the community detection algorithm on intermediate super-networks of detected communities. Both HOC and OSLOM are multi-resolution community detection methods as they have a freely tunable resolution parameter which allows them to identify communities at varying levels of resolutions, thus forming a community structure hierarchy.

### 1.2 Community Evolution in Dynamic Networks

Another major challenge for community analysis lies in the dynamic and evolving nature of online social

Sajid Yousuf Bhat is a Research Scholar at the Department of Computer Science, Jamia Millia Islamia, New Delhi-25, India (E-mail: s.yousuf.jmi@gmail.com)

Muhammad Abulaish (corresponding author) is an Associate Professor and Head of the Department of Computer Science, Jamia Millia Islamia, New Delhi-25, India (E-mail: abulaish@ieee.org)

networks with time, as most often i) new members join the network, ii) existing members leave the network, and iii) members establish/break ties and/or change intensity/weight of interactions with other members. Consequently, all these evolutionary events result in *birth*, *growth*, *contraction*, *merge*, *split*, and *death* of communities with time. Therefore, it is desirable to identify the structural changes that occur in community structures of a social network when nodes and links are added, removed, or modified.

According to the formulation of the dynamic community detection problem in [10] and [11], social interactions of certain individuals of a network are observed along a discrete time-scale, resulting in several subgraphs at each time-step. Based on these subgraphs, communities and their developments over time are identified in such a way that most interactions are explained by the inferred community structures. However, the bipartite mapping of communities for two subgraphs in these methods assumes either zero-to-one or one-to-one mapping between nodes, and hence they do not identify *merge* or *split* events. Alternatively, Greene et al. [12] proposed a heuristic threshold-based method which allows many-to-many mappings between communities across different time-steps, thus enabling the detection of *merge* and *split* events. In [9], the authors proposed *OSLOM* which starts from any initial partition from the previous state of a network and uses it for a closer analysis of communities at next time-step to facilitate community evolution tracking.

One of the major issues associated with the methods mentioned above is the determination of an optimal similarity threshold value. To overcome this problem, Wang et al. [13] proposed an approach in which important (core) nodes of communities are determined on the basis of their centrality values in the network, and mapping of communities is performed on the basis of the common core nodes between a pair of communities at different time-steps. However in this approach, all possible community pairs of two consecutive time-steps need to be considered and checked against each other.

As pointed out in [14], most of the dynamic community detection methods have a common limitation to study community identification and community evolution problems separately. In other words, communities are first identified at different time-steps and then similarities among them at different network states are established to explain their evolutionary relations. However, it is possible that community structures at two different time-steps vary drastically due to undesirable changes in the network and thus make it difficult to explain meaningful evolutionary relations between them. Therefore, it is desirable to analyze communities and their evolution in a unified manner where community structure itself provides evidence about its evolution. To this end, Cazabet et al. [15]

proposed a robust overlapping community detection method, *iLCD*, which adapts an initially detected community structure to track the changes occurring in a dynamic network. However, it considers only the addition of new edges and nodes and identifies *merge*, *growth*, and *birth* events for community evolution. Similarly, Lin et al. [14] proposed *FaceNet*, which adapts an evolutionary clustering algorithm to identify community sequences with temporal smoothness. However, the main limitations of *FaceNet* is the requirement for specifying the number of communities manually, and low scalability due to large number of matrix computations involved. Similarly, *AFOCS* [16] adapts a previous community structure to the dynamic changes in a network, including removal of nodes and edges. However, it complicates the process by defining a number of distinct actions and cases to be considered for addition and removal of nodes and edges in a dynamic network.

An adaptive density-based method, *DENGRAPH-IO*, for finding overlapping communities in dynamic-weighted networks is presented in [17]. However, some limitations of *DENGRAPH-IO* include the requirement of two input parameters ( $\epsilon$ ) and ( $\mu$ ) for defining the community density, works only for weighted networks, provides no efficient method to explicitly map communities at two consecutive snapshots of a network to define evolutionary relations between them (if any), and it requires an ageing function (often difficult to determine) to remove old interactions from the network.

### 1.3 Our Contributions

In this paper, we propose a unified framework, *HOCTracker*, which exploits a novel density-based approach to track the evolution of hierarchical and overlapping community structures in dynamic social networks. It is an extended and much improved version of our earlier works [18] and [19]. *HOCTracker* is generalized to identify community evolution by detecting *birth*, *death*, *merge*, *split*, *growth*, and *shrink* events of communities in an evolving network. It aims to address most of the issues related to the *DENGRAPH-IO* and provides a simple and efficient solution to track the evolution of overlapping communities in dynamic social networks. Moreover, the proposed framework is applicable to directed/undirected and weighted/unweighted networks. In summary, besides the novel features of *CMiner* presented in our previous paper [19], *HOCTracker* possesses the following unique features:

- Unlike most of the dynamic community detection methods, *HOCTracker* adapts a preliminary community structure (identified through a novel density-based overlapping community detection approach) to the changes occurring in a network and processes only active nodes for the new time-

step, instead of processing all nodes for every new time-step.

- It presents the design of an intermediate evolution log and maintains while adapting communities, which later on simplifies the task of identifying evolutionary mappings between the community structures of two consecutive time-steps.
- It provides heuristics to automatically determine a good approximation for the resolution parameter ( $\eta$ ), which is also used to identify hierarchical structure of overlapping communities.

Table 1 presents a summary of the distinguishing characteristics of HOCTracker against some of the recently proposed state-of-the-art methods, and it can be observed that HOCTracker aims to address all important issues related to the community analysis problem.

TABLE 1: Distinguishing characteristics of HOCTracker

	Community Overlap	Community Evolution	Community Hierarchy	Edge Weights	Outliers
HOCTracker	✓	✓	✓	✓	✓
AFOCS[16]	✓	✓	×	×	✓
CFinder[3]	✓	×	✓	✓	✓
SLPA[6]	✓	×	✓	×	×
OSLOM[9]	✓	×	✓	✓	✓
COPRA[7]	✓	×	×	×	×
MOSES[4]	✓	×	×	×	✓
SHRINK[5]	✓	×	×	✓	✓
iLCD[15]	✓	✓	×	×	✓
GraphScope[20]	×	✓	×	×	×

The rest of the paper is organized as follows. Section 2 presents the design of HOCTracker. Sections 3 and 4 present the community evolution tracking process. The issue of identifying hierarchical communities is discussed in section 5. Section 6 presents experimental results, followed by a conclusion in section 8.

## 2 PROPOSED FRAMEWORK

The proposed framework, HOCTracker, follows a density-based approach to identify communities in a social network. For density-based clustering methods (e.g., DBSCAN[21]), a cluster is searched by detecting the neighborhood of each object in the underlying database. The neighborhood of an object  $p$  relies on the distance between  $p$  and other objects in the underlying database, wherein an object  $q$  belongs to the neighborhood of an object  $p$  if the distance between  $p$  and  $q$  is less than or equal to a threshold ( $\epsilon$ ). In a graph-based context, however, a node  $q$  belongs to the neighborhood of a *directly connected node*  $p$  only if the (structural) distance between  $p$  and  $q$  is less than or equal to a threshold ( $\epsilon$ ). If the neighborhood of a given radius ( $\epsilon$ ) of an object  $p$  contains more than  $\mu$  objects, a new cluster with  $p$  as a core object is created. The process then iterates to find density-reachable objects from the core objects and defines a density-connected cluster as a maximal set of density-connected nodes, which is illustrated in figure 1. The main drawback of

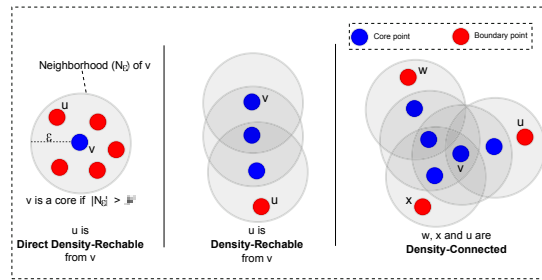


Fig. 1: A density-connected cluster

traditional density-based community detection methods is the requirement of two input parameters – a global neighborhood threshold ( $\epsilon$ ) and a minimum cluster size ( $\mu$ ). Therefore, HOCTracker does not require the global neighborhood threshold parameter ( $\epsilon$ ) to be set manually at the beginning of the process. Instead, it uses a local representation of the neighborhood threshold which is automatically calculated for each node locally. Moreover, a local version of  $\mu$  is also computed for each node automatically based on an input resolution parameter ( $\eta$ ), which can be tuned as required or can be estimated using a heuristic approach presented as Appendix B in the supplementary document of this paper.

### 2.1 Preliminaries

An important component of density-based community detection methods is a distance function used to decide whether a pair of nodes can belong to the same community or not. The commonly used distance/similarity functions include Jaccard coefficients, Cosine similarity, and topological overlap which only consider the undirected and unweighted nature of networks. For HOCTracker, we define a novel dual-layered distance function which is generalized for each directed/undirected and weighted/unweighted networks. Considering the social network as a graph  $G = (V, E_w)$ , where  $V$  is the set of nodes representing users and  $E_w \subseteq V \times V$  is the set of weighted links between the users, the proposed distance function is defined as follows.

2.1.0.1 Layer-1: It should be noted that the distance is measured only between those node pairs that are directly linked and have reciprocating interactions between them as they are expected to be less distant (more similar) than otherwise. In case of undirected networks, each link is considered to be reciprocating by treating it as a set of two oppositely directed links with the same weight as the original link. Given these considerations, the first layer of the distance function for two reciprocating nodes  $p$  and  $q$  is represented by equation 1, where  $V_p$  and  $V_q$  are sets of nodes to which nodes  $p$  and  $q$  have outgoing links/interactions respectively, and  $V_{pq}$  is the set of common nodes to which both  $p$  and  $q$  have outgoing links in the

network.

$$\text{dist}(p, q) = \begin{cases} \Delta(p, q) & \text{if } |V_{pq}| > (\eta \times \min(|V_p|, |V_q|)) - 1 \\ 1 & \text{otherwise} \end{cases} \quad (1)$$

In equation 1,  $\Delta(p, q)$  represents the *layer-2* of the distance function which is discussed shortly and  $\eta$  ( $0 < \eta \leq 1$ ) is an input parameter which specifies the resolution at which communities need to be identified. In simple terms, the first layer of the distance function ensures that the distance between two reciprocating nodes  $p$  and  $q$  is computed at the second layer only if the fraction of their commonly interacted nodes forms a significant fraction of the total outreached nodes of either  $p$  or  $q$ , i.e.,  $\min(|V_p|, |V_q|)$ . Otherwise, the distance between  $p$  and  $q$  is taken as 1 (maximum).

**2.1.0.2 Layer-2:** The second layer of the distance function takes the intensity of interactions between nodes (link weights) into consideration. It is based on the assumption that if a node  $p$  has outgoing links (interactions) to a node  $q$  and a set of nodes  $V_{pq}$  (to which  $q$  also has outgoing links) then the similarity/distance between  $p$  and  $q$  can be measured in terms of the amount of response from  $q$  and nodes in  $V_{pq}$  to the interactions of  $p$  and vice versa. Formally, the response of node  $q$  and the nodes in  $V_{pq}$  to the interactions of node  $p$  is measured as the average of the amount of per-node reciprocated interactions (edge weights) of  $q$  and the nodes in  $V_{pq}$  towards  $p$ , represented by  $\delta(p, q)$ , as given in equation 2, where  $I_{\overleftarrow{pq}}$  represents the amount of reciprocated interactions (weight) between two nodes  $p$  and  $q$ , i.e., minimum of the amount of interactions from  $p$  to  $q$  and  $q$  to  $p$ .

$$\delta(p, q) = \begin{cases} \left( \frac{\sum_{s \in V_{pq}} (I_{\overleftarrow{ps}}) + I_{\overleftarrow{pq}}}{|V_{pq}| + 1} \right) & \text{if } I_{\overleftarrow{pq}} > 0 \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

Finally, the symmetric distance between two nodes  $p$  and  $q$ ,  $\Delta(p, q)$ , is taken as the maximum of their mutual directed-response (or *minimum of the reciprocals of their mutual directed-response*) values normalized by their respective total weight of outgoing interactions (represented by  $I_{\overrightarrow{p}}$  and  $I_{\overrightarrow{q}}$  respectively) in the interaction graph, as given in equation 3. The dual-layer dis-

$$\Delta(p, q) = \begin{cases} \min \left( \frac{\delta(p, q)^{-1}}{I_{\overrightarrow{p}}}, \frac{\delta(q, p)^{-1}}{I_{\overrightarrow{q}}} \right) & \text{if } \delta(p, q) > 0 \wedge \delta(q, p) > 0 \\ 1 & \text{otherwise} \end{cases} \quad (3)$$

tance function thus defined measures the amount of maximum average reciprocity among two nodes  $p$  and  $q$  and their common neighbors, provided the overlap of their neighbors is significant. Smaller values of  $\Delta(p, q)$  represent higher response between nodes  $p$

and  $q$  and translates to more closeness between  $p$  and  $q$ .

Given a distance measure, we need to specify a neighborhood threshold to mark the boundary for any given node as required by density-based methods. However, instead of manually specifying the threshold value, we determine a *local neighborhood threshold* for a node  $p$  as the average per-receiver reciprocated interaction score of  $p$  with all its outreached neighbors. Formally, the local neighborhood threshold of a node  $p$  ( $\varepsilon_p$ ) is defined using equation 4, where  $V_p$  represents the set of nodes to which  $p$  has out-links,  $I_{\overleftrightarrow{p}}$  represents the number of reciprocated interactions of a node  $p$  (i.e.,  $\sum_{q \in V_p} \min(I_{\overleftarrow{pq}}, I_{\overrightarrow{qp}})$ , where  $I_{\overleftarrow{pq}}$  represents the number/weight of interactions from node  $p$  to node  $q$ , and  $\frac{I_{\overleftrightarrow{p}}}{|V_p|}$  represents the average number of reciprocated interactions between  $p$  and all other nodes in  $V$  to which  $p$  has out-links. The denominator  $I_{\overleftrightarrow{p}}$  represents the total count of outgoing interactions of node  $p$  and it normalizes the value of  $\varepsilon_p$  in the range  $[0, 1]$ .

$$\varepsilon_p = \begin{cases} \left( \frac{I_{\overleftrightarrow{p}}}{|V_p|} \right)^{-1} & \text{if } |V_p| > 0 \wedge I_{\overleftrightarrow{p}} > 0 \\ 0 & \text{otherwise} \end{cases} \quad (4)$$

Based on the distance function  $\text{dist}(p, q)$  and local neighborhood threshold  $\varepsilon_p$ , we define a *local  $\varepsilon_p$ -neighborhood* of a node  $p$  as the subset of  $p$ 's out-linked nodes (i.e.,  $V_p$ ) with which its distance is less than or equal to  $\varepsilon_p$ . Formally, the local  $\varepsilon_p$ -neighborhood of a node  $p$  can be defined using equation 5.

$$N_p = \{q : q \in V_p \wedge \text{dist}(p, q) \leq \varepsilon_p\} \quad (5)$$

In simple terms, we can state that  $N_p$  contains those neighbors of  $p$  in the network that have a significant topological overlap with  $p$  and an above-average interaction intensity with  $p$  in the network neighborhood. This approach thus aims to find areas of high structural density (than the surrounding) to constitute a community.

Formally, for a given resolution fraction ( $\eta$ ), a density-based community is realized by the following two key definitions.

**Definition 1 (Core node).** A node  $p \in V$  having non-zero reciprocated interactions with any of its neighbor(s) in  $V_p$  is defined to be a *core node* if its local  $\varepsilon_p$ -neighborhood contains at least  $\mu_p$  (local minimum-number-of-points threshold for  $p$ ) of nodes in  $V_p$ , as given in equation 6, where  $\mu_p = \eta \times |V_p|$ .

$$\text{CORE}_\eta(p) \Leftrightarrow |N_p| \geq \mu_p \quad (6)$$

**Definition 2 (Mutual-cores).** Two nodes  $p, q \in V$  are *mutual-cores* if both  $p$  and  $q$  are core nodes, and  $p$  belongs to local  $\varepsilon_q$ -neighborhood and  $q$  belongs to local  $\varepsilon_p$ -neighborhood.



The basic aim is to find all maximal sets of connected core nodes such that for each pair of nodes in a maximal set there exists a chain of nodes  $v_1, v_2, \dots, v_n$  such that  $v_i$  and  $v_{i+1}$  are mutual-cores for all  $i$  ranging from  $1, 2, \dots, n-1$ . The set of all such connected core nodes forms the *mutual-core connected maximal subgraph* (MCMS) of a community. A community is then defined as the union of an MCMS (backbone of the community) and local  $\varepsilon_p$ -neighborhoods of each core node  $p$  in the MCMS. The set of all such possible communities identified forms the community structure of an underlying network.

## 2.2 Finding Initial Community Structure

To find overlapping community structures from an initial snapshot  $\aleph^t$  of a dynamic network for a given resolution  $\eta$ , initially all nodes of the network are marked un-labeled and un-visited. The process randomly selects an un-visited node, say  $p$ , to find the primary community of  $p$  by determining whether it is a core node, and whether it *initiates* a new community, *joins* an existing community, or causes more than one existing communities to *merge*. To this end, local  $\varepsilon_p$  threshold for  $p$  is calculated using equation 4. If  $\varepsilon_p$  is greater than zero, then the dual-layer distance function  $dist(p, q)$  is used to determine the  $N_p$  of  $p$ . Finally, if node  $p$  qualifies as a core node then the following steps are followed to find overlapping communities:

- 1) Determine a set  $V$  of visited nodes in  $N_p$  with which  $p$  has mutual-core relations and a set  $U$  of un-visited nodes in  $N_p$ .
- 2) If  $V$  is empty, then  $p$  along with all the nodes in its  $N_p$  form a new community and their respective community lists are appended with a new community ID  $C$ . A node is assigned to a new community irrespective of its previous community allotments, thus allowing a node to belong to multiple communities. Here, node  $p$  is called a primary core of community  $C$ , and community  $C$  is called the primary community of  $p$ . Any core nodes in  $N_p$  that are not in the set  $V$  are called secondary core nodes of  $C$ , and community  $C$  forms their secondary community.
- 3) If  $V$  is non-empty and all core nodes in  $V$  have the same primary community  $C$  then  $p$  also forms a primary core of community  $C$  and the respective community lists of all the nodes in  $N_p$  including  $p$  are appended with the community label  $C$ .
- 4) If  $V$  is non-empty and some core nodes in  $V$  have different primary communities, then the primary communities of the nodes in  $V$  are merged to form a single community. The community IDs of the merged communities for all visited density-reachable nodes of  $p$  are replaced with a new community ID. The primary core nodes of the

---

### Algorithm 1: AdaptCS( $\aleph^{t+1}, C_\eta, \eta', S$ )

---

```

/*  $\aleph^{t+1}$  is the social network state at time
 $t+1$  */
/*  $C_\eta$  is the community structure identified by
HOCTracker at a particular value of  $\eta$  on the
network state  $\aleph^t$  */
/*  $\eta'$  is the value of  $\eta$  at which new community
structure is to be determined */
/*  $S$  is the set of all candidate nodes whose
neighborhood re-computation could result in
the change of the known community structure */
1 begin
2   Mark all nodes of  $S$  in the social network  $\aleph^{t+1}$  as
   un-visited;
3   foreach Remaining un-visited node  $p$  in the network do
4     enqueue( $p$ ); /* Standard insert operation on a
   Queue */
5     while Queue is not empty do
6        $p \leftarrow$  dequeue(); /* Standard removal
   operation on a Queue */
7        $p.waiting \leftarrow$  false;
8        $N_p^t \leftarrow N_p$ ; /* Save  $p$ 's current local
   neighborhood */
9       Re-compute the local neighborhood of  $p$  at  $t+1$ ,
   i.e.,  $N_p^{t+1}$ , based on  $\aleph^{t+1}$ ;
10       $N_p \leftarrow N_p^{t+1}$ ; /* Set the new
   neighborhood of  $p$  as its current
   neighborhood */
11      Compare  $N_p^t$  with  $N_p^{t+1}$  to check the following
   conditions;
12      if  $p$  emerges as a new core node then
13        Determine the set  $S$  of visited mutual-core
   nodes of  $p$  from  $N_p^{t+1}$ ;
14        Birth_Expand_Merge( $p, S$ );
15      else if primary core node  $p$  of a community  $C$  loses
   its core property then
16        Determine the set  $S$  of mutual-core nodes of
    $p$  from  $N_p^t$ ;
17        Death_Shrink_Split( $p, S, C$ );
18      else
19        if Core node  $p$  gains a set of nodes  $\vee \exists$  a visited
   mutual-core node of  $p$  in  $N_p^{t+1}$  which has a
   different primary community than  $p$  then
20           $S \leftarrow$  Visited mutual-core node of  $p$  in
    $N_p^{t+1}$  which have a different primary
   community than  $p$ ;
21          Add  $p$  to  $S$ ;
22          Birth_Expand_Merge( $p, S$ );
23        end
24        if Primary core node  $p$  of community  $C$  loses a
   set of nodes then
25           $S \leftarrow$  Mutual-core nodes of  $p$  from  $N_p^t$ 
   which are not in  $N_p^{t+1}$ ; /* Here it
   should be noted that to find
   the old mutual-core nodes of
    $p$ , the current neighborhoods
   of the nodes in  $N_p^t$  are
   considered */
26          Add  $p$  to  $S$ ;
27          Death_Shrink_Split( $p, S, C$ );
28        end
29      end
30      Mark  $p$  as visited;
31      foreach un-visited node  $q$  in  $N_p$  do
32        if not  $q.waiting$  then
33          enqueue( $q$ ); /* Standard insert
   operations on a Queue */
34           $q.waiting \leftarrow$  true;
35        end
36      end
37    end
38  end
39 end

```

---

merged communities along with node  $p$  form the primary core nodes of the new merged community, and the community lists of each node in  $N_p$  is appended with the new community ID.

- 5) Mark node  $p$  as visited.
- 6) For each node  $q$  in  $U$ , if  $q$  is not marked as *waiting* then mark it as *waiting* and add into the queue.
- 7) Repeat steps 1-6 for each node removed from the queue until it is empty.

In case node  $p$  does not qualify as a core node, it is simply marked as visited (it may be added as a non-core node for some other communities in later iterations). Steps 1–7 are repeated for each remaining un-visited node in the network. This process is generalized for dynamic networks and formally presented as algorithm 1. To find community structures from an initial snapshot  $\mathbb{N}^t$  of a dynamic network (or a static network) for a given resolution  $\eta$ , algorithm 1 is called as  $AdaptCS(\mathbb{N}^t, \emptyset, \eta, S)$  where  $S$  is the set of all nodes in the given network state, and  $\emptyset$  represents a previously empty community structure.

Real-world social networks often contain *noise* or *outliers* (i.e., nodes that do not belong to any community) and *hubs* (i.e., nodes that do not belong to a particular community but connect multiple communities and thus play an important role in information brokerage and diffusion within a network and across communities). Therefore, after finding all possible communities from an underlying network, HOCTracker labels an un-clustered/un-labeled node as a hub if it has out-going edges to the primary core nodes of more than one community, and the remaining nodes are marked as outliers. However, in this paper, we have not focused on detecting hubs from underlying networks.

### 3 TRACKING EVOLUTIONARY EVENTS

To track the evolving community structures in a dynamic network, HOCTracker first finds a preliminary community structure from an initial state of the network using the method discussed in section 2.2. Then for each new state of the network, it identifies the nodes that have caused the network to change from its earlier state to the new state. These nodes are called *active nodes* as they represent the nodes among which edge changes occur during a time-period. The following types of edge changes can occur in a network:

- 1) Addition of some edge(s) between a pair of nodes (either, both, or none of them can be a newly added node)
- 2) Removal of some edge(s) between a pair of nodes
- 3) Removal of all edges between a deleted (removed) node and its previous neighbors.

The active nodes and their respective neighbors (together called *candidate nodes*) need to be re-processed to detect and track community evolution induced

---

#### Algorithm 2: Tracker( $\mathbb{N}^t, \mathbb{N}^{t+1}, \eta$ )

---

```

/*  $\mathbb{N}^t$  and  $\mathbb{N}^{t+1}$  are social network states at
time  $t$  and  $t+1$ , respectively */
/*  $\eta$  is the resolution at which community
structures need to be identified */
1 begin
2    $S \leftarrow$  Set of all nodes in the network state  $\mathbb{N}^t$ ;
3    $C_\eta^t \leftarrow$  AdaptCS( $\mathbb{N}^t, \emptyset, \eta, S$ ); /* Identify initial
community structure */
4   Compare network states  $\mathbb{N}^{t+1}$  and  $\mathbb{N}^t$  to identify the set
 $A$  of active nodes ;
5    $S \leftarrow \emptyset$ ;
6   foreach node  $q \in A$  do
7     Add immediate neighbors of  $q$  to  $S$ ; /* i.e.,
node with which  $q$  has an edge in the
network state  $\mathbb{N}^{t+1}$  */
8   end
9    $S \leftarrow S \cup A$ ; /* Set of candidate nodes */
10   $C_\eta^{t+1} \leftarrow$  AdaptCS( $\mathbb{N}^{t+1}, C_\eta^t, \eta, S$ ); /* Call to adapt
previous community structure around the
candidate nodes on new state  $\mathbb{N}^{t+1}$  of the
network */
11   $G \leftarrow$  Map(Log); /* Log is the transition log
maintained by AdaptCS */
/*  $G$  is the bipartite graph which now
represents the evolutionary relations
between the community structures  $C_\eta^t$  and
 $C_\eta^{t+1}$  */
12 end

```

---

in the community structure by a timely change in the underlying network. Re-processing involves re-determining the local  $\varepsilon$ -neighborhood of each candidate node and comparing it to its previous state to identify any change in the memberships. Any change in the local  $\varepsilon$ -neighborhood of any candidate node considering the new state of the network can also result in a change in its local community structure. The reason behind the consideration of the neighbors of the active nodes is that in HOCTracker the local  $\varepsilon_p$ -neighborhood of a node is also dependent on the interaction behavior of its neighbor in a network. So if a node  $p$  interacts with some other node  $q$ , besides re-determining the local  $\varepsilon_p$ -neighborhoods of  $p$  and  $q$ , we need to re-determine the local  $\varepsilon$ -neighborhoods of all the neighbors of  $p$  and  $q$  to detect the induced changes by the active nodes  $p$  and  $q$ . This feature allows HOCTracker to detect all possible evolutionary events (birth, death, merger, split, shrinkage, and growth) on a growing-only network (involving node/link additions only) without using an ageing function to remove old interactions, which is often challenging to determine for other related methods.

For an evolving network, the candidate nodes can be determined either by considering a live stream of changes, i.e., a node  $p$  and its neighbors are marked as candidate nodes as soon as  $p$  causes a change (online processing); or by observing the network over a time-window to determine the set of nodes inducing some change to the network which are then re-processed later (offline processing). In either case, edges and nodes are added/removed to/from the previous state

of the network to form a new time-step network, and the candidate nodes are re-processed to identify the changes in their local  $\varepsilon$ -neighborhoods. This approach (implemented in algorithm 2) is better than other approaches wherein a re-clustering of the whole network is required at each given time-step, as in our case we only need to consider the active nodes and their direct neighbors that are comparatively less in sparse real-world networks. It should be noted that each new state of the network and its final resulting community structure form the base-line network and community structure, respectively for the next state of the network on which the new changes are expected to occur. Based on the local  $\varepsilon$ -neighborhoods and core node properties of the candidate nodes at a particular time, HOCTracker models the community-related evolutionary events as summarized below. Algorithm 1 formally presents the implementation of the evolutionary events tracking process. A detailed description about the various cases highlighted here is provided in Appendix A, appearing as a supplemental material to this paper.

- *A new core node emerges:* In this case, a previous non-core node (including an outlier or a newly added node) in the network becomes a core node. Such a new core node  $p$  may either join the MCMS of an existing community causing its *expansion*, form a totally new community (*birth*), or may cause two communities to *merge* by joining their respective MCMSs through its  $N_p$ . These situations are generalized and handled by algorithm 3.
- *A core node becomes a non-core:* In this case, an existing core node loses its core node property due to some change(s) in its neighborhood. Such a situation could either *split* a community due to a cut in its MCMS resulting from a lost core node, or *shrinkage* of a community. These situations are generalized and handled by algorithms 4 and 5.
- *A core node gains nodes (or mutual-core relations) and/or loses nodes but remains a core:* In this case, the gain or loss of nodes from the local neighborhoods of core nodes can either result in a *merge* and *growth* (for gain of nodes), and/or *shrink* and *split* (for loss of nodes) of communities in the evolving network. These situations are handled by algorithms 3 (for gain), and 4 and 5 (for loss).

## 4 MAPPING COMMUNITIES ACROSS TIME-STEPS

HOCTracker follows a node-based incremental approach to track community-related evolutionary events wherein one (possible candidate) node is considered at a time and changes induced by it in the network are accordingly incorporated in the resulting community structure, somewhat similar to DENGGRAPH-IO [17]. This approach results in many

---

### Algorithm 3: Birth\_Expand\_Merge( $p, S$ )

---

```

/*  $p$  is the new core node or an existing core
   which gained new nodes and/or mutual-core
   relations */
/*  $S$  is the set of visited nodes with which  $p$ 
   established new mutual-core relations (in case
    $p$  is not a new core node then  $S$  also includes
    $p$ ) */
1 begin
2   if  $S$  is empty  $\wedge$   $p$  has no primary community then
3     Node  $p$  causes the birth of a new community to
       which  $p$  forms a primary core node; /* Log entry
       for birth of new community is also made */
4   else if  $S$  includes nodes that have the same primary
       community  $C$  then
5     if  $p$  does not have a primary community then
6        $p$  is also made a primary core of community  $C$ ;
       /* causing the growth of the
       community  $C$  */
7     end
8   else if  $S$  includes nodes that have different primary
       communities then
9     The distinct primary communities of the nodes in  $S$ 
       are merged to form a new community; /* Log
       entry for the merge of communities in  $S$ 
       is also made */
10    The merging community IDs assigned to the nodes
       are replaced by a new community ID;
11    Node  $p$  also forms a primary core node of the new
       merged community;
12  end
13   $O \leftarrow N_p^{t+1} - S$ ;
14  foreach  $q \in O$  which is not labeled with the primary
       community of  $p$  do
15    The ID of the primary community of  $p$  is appended
       to the community list of  $q$ ; /* causing the
       growth of the primary community of  $p$  */
16  end
17 end

```

---

intermediate evolution graphs for the same underlying network, depending on the order of processing of the candidate nodes<sup>1</sup>. For example, figure 2 shows the sequences of changes induced in the same initial community structure of a particular network by processing the same candidate nodes, but in a different order. Each encircled alphabet in the figure represents a community and a labeled directed edge represents the transition event induced in the community, at its source, on re-computing the neighborhood of some candidate node, resulting in the community at its destination. Both the sequences start with the initial two communities  $A$  and  $B$  and finally end up with three same communities, however the intermediate transitions are different.

As mentioned earlier, DENGGRAPH-IO [17] does not provide a solution to map the evolutionary relations between communities across a time-step. Though, a simple approach to find the mappings between the communities at two consecutive time-steps could be to define a threshold of overlap to determine the

1. It should however be noted that irrespective of the order of processing of the candidate nodes, the final resulting community structure is the same for a particular state of the network

**Algorithm 4: Death\_Shrink\_Split( $p, S, C$ )**

```

/*  $p$  is the node which lost its core property
or an existing core which loses nodes from its
local neighborhood */
/*  $S$  is the set of nodes with which  $p$  had
mutual-core relations at time  $t$ , (If  $p$  is still
a core node at  $t+1$ , then  $S$  also includes  $p$ ) */
/*  $C$  is the primary community of  $p$  */

1 begin
2   if  $S$  is empty then /* Log entry for the death of
community  $C_i$  is also made */
3     Remove the community ID of  $C$  from node  $p$  and
each node in its  $N_p^t$ ;
4   end
5   else
6     Mark each node in  $S$  as un-crawled;
7      $k = 1$ ;
8     while  $\exists q \in S$  which is un-crawled do
9       Select an un-crawled node  $q \in S$ ;
10       $W_k \leftarrow \text{Crawl}(q)$ ;
11      Replace the community ID  $C$  from all nodes in
 $W_k$  with a new community ID;
/* More than one crawls (i.e.,  $k > 1$ )
means a split of community  $C$  wherein
log entries are made accordingly */
12       $k++$ ;
13    end
14     $O = N_p^t - S$ ;
15    Add  $p$  to  $O$ ;
16    foreach  $u \in O$  which is still labeled with the community
ID  $C$  do /* these nodes are not retained
by any new form of community  $C$  */
17      Remove the ID  $C$  from  $u$ ; /* In case a
single crawl resulted earlier, then
a log entry for the shrinkage of
community  $C$  is made */
18    end
19  end
20 end

```

**Algorithm 5: Crawl( $p$ )**

```

/*  $p$  is a core node from which a BFS crawl has
to be started to find the set of all
density-reachable nodes from  $p$  */

1 begin
2    $C \leftarrow \emptyset$ ; /* The crawl is initially empty */
3   enqueue( $p$ ); /* A standard insertion into a
Queue */
4   while Queue is not empty do
5      $p = \text{dequeue}()$ ; /* Standard removal from a
Queue */
6      $S \leftarrow$  All mutual-core nodes of  $p$ ; /* The current
local neighborhood of node  $q$ , i.e.,  $N_q$ 
is considered for the crawl */
7     foreach  $u \in S \wedge u \notin C$  do
8       enqueue( $u$ );
9     end
10    Add  $p$  to  $C$ ;
11    Mark  $p$  as crawled;
12  end
13   $T \leftarrow \emptyset$ ;
14  foreach  $v \in C$  do
15     $T \leftarrow T \cup N_v$ ; /* Where  $N_v$  is the current
local neighborhood of node  $v$  */
16  end
17   $C \leftarrow C \cup T$ ;
18  return  $C$ ;
19 end

```

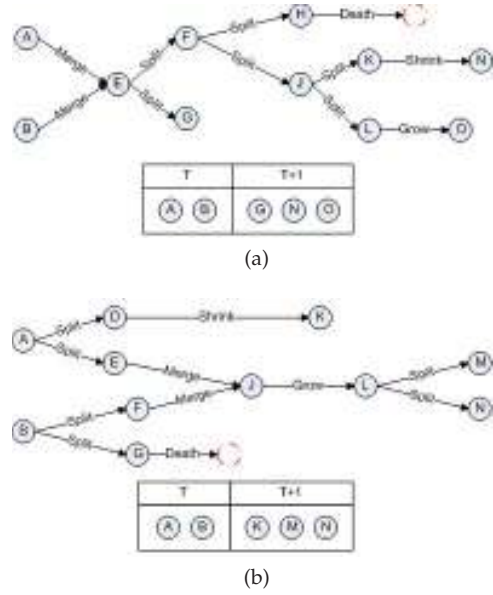


Fig. 2: Community transitions after processing nodes in different orders

similarity between two communities across a time-step [22], deciding an appropriate level of overlap may often be difficult. To this end, HOCTracker maintains an intermediate evolution log (IEL) that records the intermediate transitions induced in the communities on processing each candidate node. For each intermediate community  $C$  resulting after re-computing the local  $\varepsilon$ -neighborhood of a candidate node at time  $t + 1$ , an entry is made in the IEL. The IEL consists of the following four fields: i) *ID*: It represents the label of the resulting community, ii) *Parents<sup>T</sup>*: It includes the list of IDs of the actual communities at time  $t$  that have resulted in this community. These parent IDs are propagated forward as the parents of any future community resulting from this community on the network state at time-step  $t+1$ , iii) *Transition*: It represents the immediate event, including the IDs of the immediate communities that resulted in this community. This field can be used to track the transitions of a particular community on processing the candidate nodes; however, for only mapping the communities across time-steps this field is optional and can be omitted, iv) *Live*: This is a flag to represent the status of the community. A value of 1 represents that the community is currently alive, whereas a 0 entry represents that the community has transitioned into some other state. For a new entry this field is set to 1 (except for a death event for which it is set to 0) until its representative community transitions to some other state, wherein the value is changed to 0.

Table 2 shows the final log entries after following the transition sequence of figure 2b. The sequence results in three final communities K, M, and N at time  $t + 1$  (represented by the entries in the final log



TABLE 2: Final Intermediate Evolution Log (IEL) for the transition sequence of figure 2b

ID	Parents <sup>t</sup>	Transition	Live
A	A	-	0
B	B	-	0
D	A	Split(A)	0
E	A	Split(A)	0
F	B	Split(B)	0
G	B	Split(B)	0
-	B	Death(G)	0
J	A,B	Merge(E, F)	0
K	A	Shrink(D)	1
L	A,B	Grow(J)	0
M	A,B	Split(L)	1
N	A,B	Split(L)	1

where the *Live* field is 1) from two initial communities A and B at time  $t$ . In order to map the evolution of communities at time  $t$  to the communities at time  $t + 1$ , we consider the final live log entries (for which *Live* field is 1) and form two sets of community IDs. One set  $S^{t+1}$  contains the IDs of the communities which are live in the log (i.e., the final communities at time  $t + 1$ ) and the other set  $S^t$  contains the IDs of the actual (parent) communities at time  $t$ . Now for each community  $C'$  in the set  $S^{t+1}$ , we form an undirected link from  $C'$  to each community  $C$  of the set  $S^t$ , which is present in the  $Parents^t$  field of the log entry for community  $C'$ , only if the communities  $C$  and  $C'$  share a primary core node with the same local  $\varepsilon$ -neighborhood. For large real-world networks consisting of hundreds and thousands of communities, this approach significantly reduces the number of comparisons to be made between communities as compared to [13], [22], [23], [24] (to identify evolutionary relations) as it involves comparing a final community at time  $t + 1$  to only those communities at time  $t$  which are present in the  $Parents^t$  field of its log entry. The resultant from this mapping scheme is a bipartite graph between the sets  $S^t$  and  $S^{t+1}$ . A possible bipartite graph for the log entries given in table 2 is shown in figure 3. This final bipartite

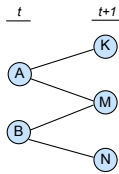


Fig. 3: A bipartite graph of community mappings across a time-step

graph explains the actual mapping and evolutionary relations between the communities at times  $t$  and  $t + 1$  as follows:

- A degree greater than 1 in the bipartite graph for a community  $C$  of time  $t$  represents a *split* of community  $C$  into the communities of time  $t + 1$ , to which  $C$  is linked to in the bipartite graph.
- A degree greater than 1 in the bipartite graph for a community  $C$  of time  $t + 1$  means that the community  $C$  has resulted from the *merge* of a

**Algorithm 6:** Map(*Log*)

---

```

/* Log is the intermediate transition log
maintained by AdaptCS for each state of the
network for logging the intermediate
transition events */
1 begin
2   St ← ∅;
3   St+1 ← ∅;
4   foreach entry i in the Log do
5     St ← St ∪ Parentit; /* The set of all
community IDs of time t */
6   end
7   E ← ∅;
8   foreach Live entry, for a community C, in the Log do
9     Add C to St+1;
10    foreach community-ID C' in the Parentt set of the log
entry do
11      if Communities C and C' share a primary core node
then
12        Form an edge e between C and C';
/* Forming the edges of a
bipartite graph between the set
St+1 of final communities at time
t+1 and the set St of communities
at time t */
13        Add e to E;
14      end
15    end
16  end
17  G ← (St, St+1, E); /* The bipartite graph */
18  return G;
19 end
  
```

---

set  $S$  of multiple communities (to which  $C$  is linked to in the bipartite graph) of time  $t$ . In case a community  $C'$  of set  $S$ , involved in a merge, has degree greater than 1 in the bipartite graph, it means that only a split part of the community  $C'$  participated in the merge to form community  $C$ . For example, in the bipartite graph of figure 3, community  $J$  has resulted from the merge of the split parts of two communities  $A$  and  $B$ .

- A community  $C$  of time  $t$ , which has no link to any community of time  $t + 1$ , represents the *death* of community  $C$  at time  $t + 1$ .
- A community  $C'$  of time  $t + 1$ , which has no link to any community of time  $t$ , represents the *birth* of community  $C'$  at time  $t + 1$ .
- Two communities across a time-step that have a link in the bipartite graph and each has a degree of 1 are checked as follows:
  - If the community  $C'$  of time  $t + 1$  has the number of nodes less than its connected community  $C$  of time  $t$ , it means that community  $C'$  has resulted from the *shrinking* of community  $C$ .
  - If the community  $C'$  of time  $t + 1$  has the number of nodes greater than its connected community  $C$  of time  $t$ , it means that community  $C'$  has resulted from the *growth* of community  $C$ .
  - If the community  $C'$  of time  $t + 1$  has the number of nodes same as that of its con-

nected community  $C$  of time  $t$ , it means that community  $C$  has suffered no change and is represented by community  $C'$  at time  $t + 1$ .

It should be noted that while mapping the communities across multiple time-steps of a dynamic network, a new evolution log is started for each new time-step  $t + 1$ . The final communities from the previous time-step  $t$  form the initial entries in the new log. For example, the first two entries in table 2 represent the final communities at time  $t$  while the log is maintained for transitions of time-step  $t + 1$  as illustrated in figure 2b. This process finally results in an  $m$ -partite graph (where  $m$  represents the number of time-steps), which gives a step-wise evolutionary mapping of communities across various time-steps of a dynamic network.

## 5 FINDING COMMUNITY HIERARCHY

HOCTracker identifies larger communities for small values of  $\eta$ , whereas larger values of  $\eta$  yield smaller communities. Thus,  $\eta$  can be tuned to detect communities at different levels of granularity, naturally forming a hierarchical community structure. This feature assigns HOCTracker to the multi-resolution class of hierarchical community detection methods.

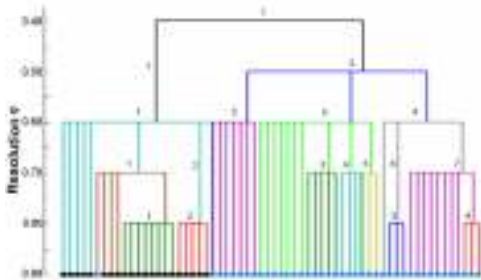


Fig. 4: Color coded dendrogram representation of the hierarchical overlapping community structure identified by HOCTracker on Dolphin network

Figure 4 presents the hierarchical overlapping community structure identified by HOCTracker on the Dolphins network [25], which reveals that the communities at  $\eta = 50\%$  almost perfectly match the ground truth (represented by leaf-node shape and color in the dendrogram). Increasing the value of  $\eta$  from 50% to 60% breaks one of the two communities into three smaller communities, thus resulting in a total of four communities with no outliers, and so on.

Unlike traditional community detection methods, HOCTracker defines the relation between community structures identified at a level  $L$  (for a smaller value of  $\eta$ ) and level  $L - 1$  (for a larger value of  $\eta$ ) in terms of evolutionary mapping between communities at two consecutive snapshots as discussed earlier. Communities at level  $L + 1$  can be viewed as resulting from the *merge* and *growth* of communities at level  $L$ , besides the *birth* of new communities, due

to the formation of new core nodes or expansion of the local neighborhoods of existing core nodes at a smaller value of  $\eta$ . We call this process as a *RollUp* operation on the communities at a given level which is formally presented as algorithm 1 in Appendix C. On the other hand, community structure at level  $L - 1$  can be viewed as resulting from the *death*, *split*, and *shrinkage* of communities at level  $L$  due to core nodes losing their core property or reduction of the local neighborhoods of sustaining core nodes at a larger value of  $\eta$ . We call this process as a *DrillDown* operation on the communities of previous level which is presented as algorithm 2 in Appendix C.

## 6 EXPERIMENTAL RESULTS

We compare HOCTracker<sup>2</sup> with some state-of-the-art community detection methods on dynamic real-world networks. The methods include overlapping community detection method MOSES [4] which has also been used for dynamic community tracking in [12]; the dynamic community detection method AFOCS [16]; and SHRINK [26]. Other methods include overlapping community detection methods COPRA [7], SLPA[6], CPM (CFinder) [3] used for detecting dynamic communities in [23], OSLOM [9], and iLCD [15]. For comparison, we have used the best results generated by each method. It should be noted that, to the best of our knowledge, no method in literature implements an explicit way (in its source-code) to track the evolution of the communities across time-steps and only gives the final community structure at the specified time-step. Therefore, we compare the methods by considering their results at different time-step graphs of a dynamic network individually and provide the evolution results for HOCTracker only.

### 6.1 Results on DBLP Co-Authorship Network

The DBLP Co-authorship network is a subset of the DBLP<sup>3</sup> dataset consisting of 2,723 vertices representing authors. An edge exists between two authors if they have written a paper together during 1990 to 2010. The dynamic network is divided into 9 time-step graphs, each depicting co-authorship relations over 5 years ([1990 – 1994][1992 – 1996][1994 – 1998][1996 – 2000][1998 – 2002][2000 – 2004][2002 – 2006][2004 – 2008][2006 – 2010]). In addition, for each time-step graph, every vertex is associated to a set of 43 attributes corresponding to the number of publications in each conference/journal during the respective time-period. The conferences/journals are grouped on the basis of their concerned areas as listed in table 3. The information about the number and venue of author

2. The binaries of the implemented algorithms along with a proper documentation are available at <http://www.abulaish.com/HOCTracker-Binaries-and-NetworkExamples.zip>

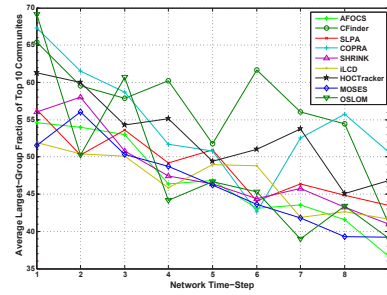
3. <http://dblp.uni-trier.de/>

TABLE 3: Publication venues of the authors in DBLP co-authorship network

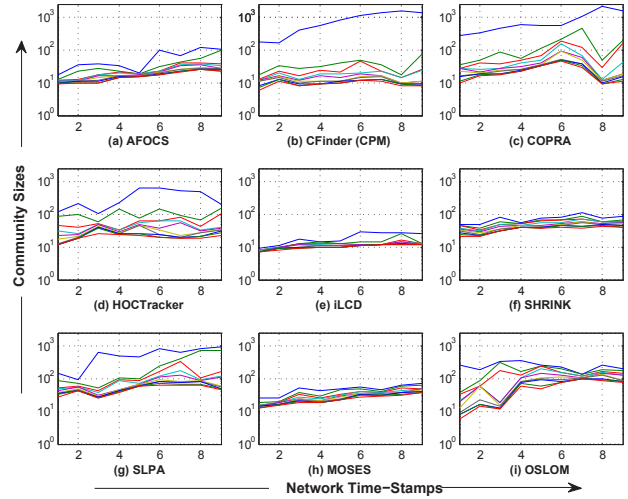
Group	Conferences/Journals
Bio-Informatics	BioInfo, BMCBio
Data Mining	DMKD, ICDM, PAKDD, KDD, TKDD, StatAnalDeMining, SIGKDDExp, IDA, SDM, SAC, WWW
Knowledge, Data and Information Systems	IEEETransKnowlDtEn, IEEEIntSys, ICDE, CIKM, KnowlInfSyst, ACMTransInfSys, JIntellInfSys, DataKnEng, InfSys, IntellDtAnal, SIGMOD, VLDB, DASFAA, DEXA, WWW
Artificial Intelligence	IJCAI, AAAI, ECML_PKDD, ICML, MachineLearning, ECAL, JMLR, JIntellInfSys, IntellDtAnal, IEEEIntSys, PatternRecog, ILP, IDA
Database Systems	PVLDB, VLDB, VLDBJ, PODS, EDBT, SIGMOD, ACMTransDBSys, DASFAA, ICDT, DEXA
General	CommunACM

publications for each time-step is used to evaluate the significance of authors' communities identified by various community detection methods. For performance comparison, we measure the fraction of nodes in each community that collectively have the maximum number of publications in the same Journal/Conference group shown in table 3. This measure, called as the maximum grouping fraction of a community, is averaged for top-10 (largest 10) communities identified by each method for each time-step graph of the dynamic DBLP network separately as shown in figure 5a. A higher value for the grouping fraction means that the identified communities contain more authors which tend to publish in related areas. On analyzing the results shown in figure 5a, it can be seen that CFinder(CPM), HOCTracker, and COPRA identify more significant grouping on the individual time-step graphs of the co-authorship network.

However, it is possible that the higher grouping score is the result of many tiny communities that contain very few authors publishing in the same areas. To further evaluate the significance of the community structures, we compare the size distribution of the top-10 communities identified by various methods on each time-step graph, as shown in figure 5b, wherein each plot line corresponds to communities of equivalent size for various time-steps. From figure 5b it can be observed that AFOCS, iLCD, SHRINK, and MOSES tend to identify small communities (less than 100 nodes) and their grouping fractions (from figure 5a) are also low. SLPA and OSLOM tend to identify larger communities, but their grouping fraction is still low. Moreover, OSLOM yields good grouping fractions for time-steps 1 and 3, but verifying it in figure 5b reveals that it is caused by the large number of tiny communities ( $\leq 10$  nodes) at these time-steps. A similar pattern is shown by CFinder as most of its identified largest communities are tiny. Moreover, the largest community (blue line in figure 5b) for CFinder appears to contain the majority of nodes of the whole network as the time-steps pass. This may not reveal an acceptable community structure of the underlying network. The single big community problem also appears to occur with COPRA, but only towards the last three time-steps. It can be thus



(a) Largest grouping fractions of top-10 communities



(b) Size distribution of top-10 communities

Fig. 5: Experimental results on dynamic DBLP co-authorship network

argued that only HOCTracker and COPRA produce communities from the DBLP network with significant real-world grouping and size distribution.

A word-cloud representation of the authors' publications for each of the top-5 (largest 5) communities identified by COPRA, CFinder, and HOCTracker on each time-step network is shown in figure 6. The size of a tag (Conference/Journal name) in each word-cloud represents the fraction of papers published by all authors in the respective community (represented by rows with decreasing community size from top to bottom) for a particular time-step (represented by columns). From figure 6 it can be argued that HOCTracker identifies the *Bio-Informatics* community earlier (in 6<sup>th</sup> time-step) than COPRA (8<sup>th</sup> time-step) and CFinder (7<sup>th</sup> time-step) as a second largest community which can be considered as a distinguishing feature.

As mentioned earlier, none of the dynamic community detection methods explicitly implements community evolution mapping. Therefore, we present evolutionary mapping results of HOCTracker among top-5 communities identified for each time-step. However, in this case, changes are added to the network at each time-step instead of considering each time-step graph



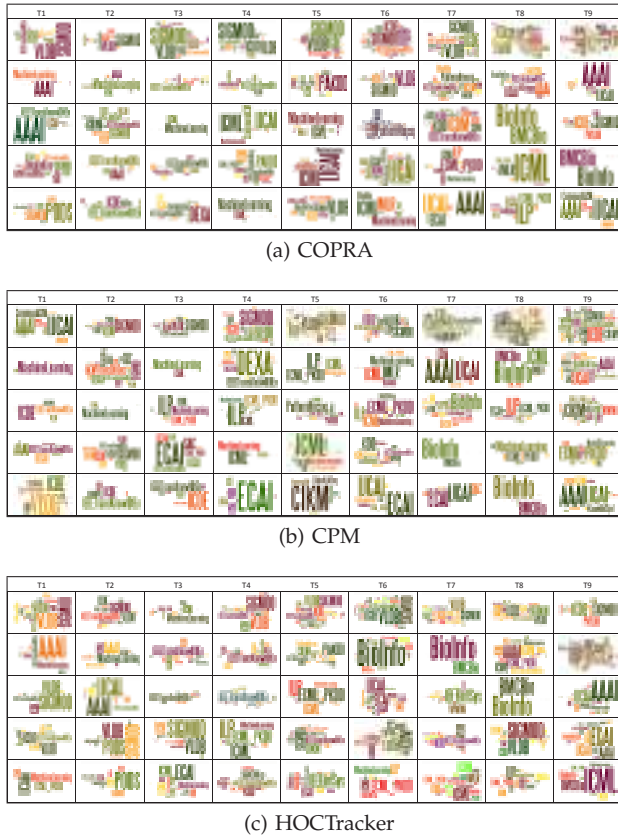


Fig. 6: Word-cloud representation of the communities identified from DBLP network

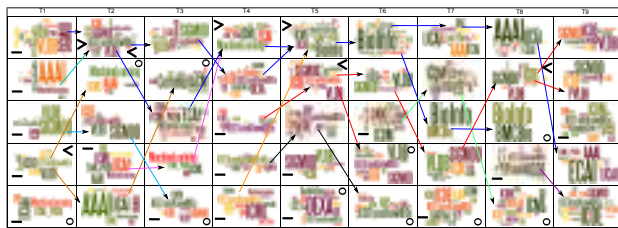


Fig. 7: Community Mapping

separately which can result in communities different than figure 6. The mapping results of HOCTracker are shown in figure 7, wherein arrows indicate evolutionary relations and the events like birth, death, merge, and split are represented by markers  $-$ ,  $o$ ,  $>$ , and  $<$ , respectively. A key observation from figure 7 is that HOCTracker identifies *Bio-Informatics* community being initiated at time-step 4 within the *Machine-Learning* community which tends to grow till time-step 6 after which it splits to form a separate community at time-step 7.

## 6.2 Wikipedia Election Network

The Wikipedia Election network [27] is a dynamic directed network of about 8000 users from the English Wikipedia who voted for and against each other in admin elections from year 2004 to 2008. Nodes represent individual users and directed edges represent

votes. Edges are positive (“for” vote) and negative (“against” vote) represented by the edge weights 1 and  $-1$ , respectively. The dataset is divided into five sub-networks based on the year of voting, and the results are only generated for HOCTracker as other methods do not consider the directed nature of networks. Starting with the network of year 2004, HOCTracker identifies the initial community structure. Then for each subsequent year, it adds the respective sub-network to the current state of the network and identifies the changes induced to the existing community structure for the new state of the network. HOCTracker finds highly overlapping communities from each state (year) of the voting network.

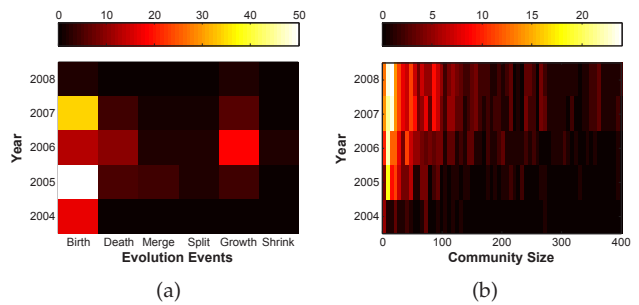


Fig. 8: Heat-map representing year-wise distribution of the number of communities and evolution events identified by HOCTracker on Wikipedia Election network

The number of various evolution events identified by HOCTracker according to the  $m$ -partite evolution graph (discussed in section 4) from the year-wise Wikipedia Election network is presented in figure 8a. In this figure, the heat-map represents the count of the various community evolution events for each year from 2004 – 2008, wherein dark color represents more events and white color represents less events as indicated in the colormap. Similarly, figure 8b shows the yearly size distribution of the communities identified by HOCTracker on the dynamic Wikipedia Election network. In figure 8b, the  $x$ -axis is a log-scale representing the community size and the colored contour lines represent the count of communities for each year, with red and blue colors representing more and less number of communities respectively as indicated in the colormap. From figure 8 it can be seen that the evolution of the network mainly involves birth, death, and growth events with birth and growth being more significant. Moreover, both the size and count of the communities is less in the beginning years but increase significantly towards the later years wherein relatively larger communities are also found. The analysis of the community evolution trend on the Wikipedia Election network reveals that each year new nodes either tend to join existing



larger communities (and cause their growth) or form completely new communities. Moreover, some young communities tend to dissolve (death) and their nodes join other existing communities (relatively older) instead of involving in merge, split, or shrinkage of communities. Such a behavior of joining a network can often be shown by the nodes (individuals) of real-world networks, wherein they tend to join a popular group existing in the network or completely form a new group within known acquaintances only, or arbitrary join and withdraw from groups until they know the network and decide their place within it. Experimental results on some additional networks can be found in Appendix E provided as the supplementary material to this paper.

## 7 COMPARISON OF RUNNING TIME

Considering time complexity, the main part of HOCTracker involves analyzing the local neighborhood of each node in the network, and for each node this cost is proportional to its out-degree. Hence, total cost for this step on a network with  $n$  nodes is  $O(deg(p_1) + deg(p_2) + \dots + deg(p_n))$ , where  $deg(p_i)$  is the out-degree of the node  $p_i$ . For a complete graph of  $n$  nodes, the degree of each node is  $n - 1$ , leading to a worst case complexity for this step as  $O(n^2)$ . However, in general, real-world networks show sparser degree distribution, resulting in an  $O(n)$  or more generally  $O(m)$  ( $m$  is the number of edges) average case complexity. In reality, HOCTracker also involves a post-merge step (detailed in Appendix D) whose complexity depends on the number of identified overlapping communities having relatively many common nodes, and the heuristics for estimating  $\eta$  whose complexity depends on the number of edges in the networks. Moreover, while tracking the evolution of communities, it also involves the sub-process of crawling the mutual-core relations of the nodes to identify splitting and re-labeling of nodes which makes it difficult to provide a true computational analysis of the method in both static and dynamic contexts.

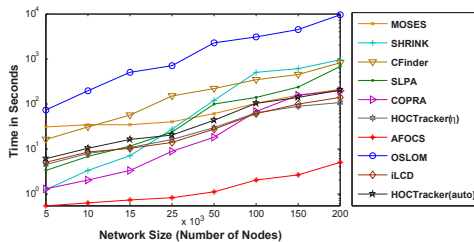


Fig. 9: Comparison of running time

Figure 9 compares the running time required by the various methods on a range of synthetic networks of different sizes. The networks used for figure 9 are the LFR-benchmarks [28] generated by varying the number of nodes from 5,000 to 200,000, with

average degree  $\langle k \rangle = 10$  and max. degree  $\langle k \rangle = 50$ . To generate the results for figure 9, we use two versions of the proposed method which include HOCTracker( $\eta$ ) (requiring the resolution value  $\eta$  as input), and HOCTracker(auto) (automatically determining a value of  $\eta$  using the heuristics given in Appendix B) along with the other methods. The results show that AFOCS runs faster than all other methods; HOCTracker( $\eta$ ), HOCTracker(auto), MOSES, COPRA, and iLCD perform comparable to each other. However, HOCTracker( $\eta$ ) runs slightly faster for larger networks. Moreover, SHRINK, CFinder, and SLPA run slower for larger networks and OSLOM takes largest time to generate the results. From these results, we conclude that, in general, the time complexity of HOCTracker is comparable to some faster methods in literature and better than some benchmark methods like OSLOM and CFinder.

## 8 CONCLUSION

In this paper, we have proposed a novel density-based framework, HOCTracker, to track community evolution in dynamic social networks. Unlike existing methods, HOCTracker adapts a preliminary community structure (identified through a novel density-based overlapping community detection approach) to the changes occurring in a network and processes only active nodes for the new time-step. It uses an efficient log-based approach to map evolutionary relations between communities identified at two consecutive time-steps of a dynamic network. Moreover, HOCTracker identifies all community evolutionary events, and it does not require an ageing function to remove old interactions for identifying these events. Experimental results have shown that community structures identified by HOCTracker on some well-known benchmark networks are significant and better than the community structures identified by the state-of-the-art methods. Moreover, the nature of dynamic communities identified from Wikipedia Election network reflects that HOCTracker can be used to identify real-world community evolution patterns.

## APPENDIX

Various appendices cited in this paper can be found in the supplementary material provided with this submission.

## REFERENCES

- [1] J. Xie, S. Kelley, and B. K. Szymanski, "Overlapping community detection in networks: the state of the art and comparative study," *arXiv preprint arXiv:1110.5813*, 2011.
- [2] S. Y. Bhat and M. Abulaish, "Analysis and mining of online social networks: emerging trends and challenges," *WIREs: Data Mining and Knowledge Discovery*, vol. 3, no. 6, pp. 408–444, 2013.

- [3] G. Palla, I. Derényi, I. Farkas, and T. Vicsek, "Uncovering the overlapping community structure of complex networks in nature and society," *Nature*, vol. 435, no. 7043, pp. 814–818, 2005.
- [4] A. McDaid and N. Hurley, "Detecting highly overlapping communities with model-based overlapping seed expansion," in *Proceedings of the 2010 International Conference on Advances in Social Networks Analysis and Mining*, ser. ASONAM '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 112–119.
- [5] J. Huang, H. Sun, J. Han, and B. Feng, "Density-based shrinkage for revealing hierarchical and overlapping community structure in networks," *Physica A: Statistical Mechanics and its Applications*, vol. 390, no. 11, pp. 2160–2171, 2011.
- [6] J. Xie and B. K. Szymanski, "Towards linear time overlapping community detection in social networks," in *Proceedings of the 16th Pacific-Asia conference on Advances in Knowledge Discovery and Data Mining - Volume Part II*, ser. PAKDD'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 25–36.
- [7] S. Gregory, "Finding overlapping communities in networks by label propagation," *New Journal of Physics*, vol. 12, no. 10, p. 103018, 2010.
- [8] P. Kumar, L. Wang, J. Chauhan, and K. Zhang, "Discovery and visualization of hierarchical overlapping communities from bibliography information," in *Proceedings of the 2009 Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing*, ser. DASC '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 664–669.
- [9] A. Lancichinetti, F. Radicchi, J. J. Ramasco, and S. Fortunato, "Finding statistically significant communities in networks," *PLoS ONE*, vol. 6, no. 5, 2011.
- [10] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan, "Group formation in large social networks: membership, growth, and evolution," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '06. New York, NY, USA: ACM, 2006, pp. 44–54.
- [11] C. Tantipathananandh, T. Berger-Wolf, and D. Kempe, "A framework for community identification in dynamic social networks," in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '07. New York, NY, USA: ACM, 2007, pp. 717–726.
- [12] D. Greene, D. Doyle, and P. Cunningham, "Tracking the evolution of communities in dynamic social networks," in *Proceedings of the 2010 International Conference on Advances in Social Networks Analysis and Mining*, ser. ASONAM '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 176–183.
- [13] Y. Wang, B. Wu, and N. Du, "Community Evolution of Social Network: Feature, Algorithm and Model," *arxiv*, vol. physics.soc-ph, 2008.
- [14] Y.-R. Lin, Y. Chi, S. Zhu, H. Sundaram, and B. L. Tseng, "Analyzing communities and their evolutions in dynamic social networks," *ACM Trans. Knowl. Discov. Data*, vol. 3, pp. 8:1–8:31, April 2009.
- [15] R. Cazabet, F. Amblard, and C. Hanachi, "Detection of overlapping communities in dynamical social networks," in *Social Computing (SocialCom), 2010 IEEE Second International Conference on*, aug. 2010, pp. 309–314.
- [16] D. Greene, D. Doyle, and P. Cunningham, "Tracking the evolution of communities in dynamic social networks," in *Advances in Social Networks Analysis and Mining (ASONAM), 2010 International Conference on*, aug. 2010, pp. 176–183.
- [17] T. Falkowski, *Community Analysis in Dynamic Social Networks*. Sierke, 2009.
- [18] S. Y. Bhat and M. Abulaish, "OCTracker: A density-based framework for tracking the evolution of overlapping communities in OSNs," in *Advances in Social Networks Analysis and Mining (ASONAM), 2012 IEEE/ACM International Conference on*, aug. 2012, pp. 501–505.
- [19] —, "A density-based approach for mining overlapping communities from social network interactions," in *Proceedings of the 2nd International Conference on Web Intelligence, Mining and Semantics*, ser. WIMS '12. New York, NY, USA: ACM, 2012, pp. 9:1–9:7.
- [20] J. Sun, C. Faloutsos, S. Papadimitriou, and P. S. Yu, "Graphscope: parameter-free mining of large time-evolving graphs," in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2007, pp. 687–696.
- [21] M. Ester, H. Kriegel, S. Jörg, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proceedings of the International Conference on Knowledge Discovery from Data*, 1996, pp. 226–231.
- [22] T. Y. Berger-Wolf and J. Saia, "A framework for analysis of dynamic social networks," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '06. New York, NY, USA: ACM, 2006, pp. 523–528.
- [23] G. Palla, P. Pollner, A.-L. Barabási, and T. Vicsek, "Social group dynamics in networks," in *Adaptive Networks*. Springer, 2009, pp. 11–38.
- [24] S. Asur, S. Parthasarathy, and D. Ucar, "An event-based framework for characterizing the evolutionary behavior of interaction graphs," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 3, no. 4, p. 16, 2009.
- [25] D. Lusseau, K. Schneider, O. J. Boisseau, P. Haase, E. Sloaten, and S. M. Dawson, "The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations," *Behavioral Ecology and Sociobiology*, vol. 54, pp. 396–405, 2003.
- [26] J. Huang, H. Sun, J. Han, H. Deng, Y. Sun, and Y. Liu, "Shrink: a structural clustering algorithm for detecting hierarchical communities in networks," in *Proceedings of the 19th ACM international conference on Information and knowledge management*, ser. CIKM '10. New York, NY, USA: ACM, 2010, pp. 219–228.
- [27] J. Leskovec, D. Huttenlocher, and J. Kleinberg, "Predicting positive and negative links in online social networks," in *Proceedings of the 19th international conference on World wide web*, ser. WWW '10. New York, NY, USA: ACM, 2010, pp. 641–650.
- [28] A. Lancichinetti and S. Fortunato, "Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities," *Physical Review E*, vol. 80, p. 016118, 2009.



interests include data mining, social network analysis, and machine learning.



He has published over 65 research papers in reputed conference proceedings and journals related to his area of interests.

**Sajid Yousuf Bhat** received the Masters degree in Computer Applications from the University of Kashmir, Srinagar, and the PhD degree in Computer Science from Jamia Millia Islamia (A Central University), Delhi in 2009 and 2014, respectively. While at Jamia Millia Islamia, his research was supported by the UGC-BSR Fellowship for meritorious students. He is currently an Assistant Professor at the Computer Science department of IP College (University of Delhi). His research

**Muhammad Abulaish** received the Masters degree in Computer Applications from the Motilal Nehru National Institute of Technology, India, and PhD degree from the Indian Institute of Technology Delhi in 1998 and 2007, respectively. He is currently an Associate Professor and Head of the Computer Science department at the Jamia Millia Islamia (A Central University), Delhi. His research interests span over the areas of data mining, web intelligence, and security informatics. He is a senior member of the IEEE, ACM, and CSI.

APPENDIX A  
TRACKING EVOLUTIONARY EVENTS

In a generalized case, after identifying a base-line community structure, for a new time-stamp network the candidate-nodes are identified, their respective local  $\varepsilon$ -neighborhoods are re-computed and their core-node properties checked on the new network state. Based on the local  $\varepsilon$ -neighborhoods and core-node properties of the candidate-nodes of a particular time, HOCTracker models the community related evolutionary events as presented in the following sub-sections.

*A. A New Core Node Emerges*

In this case, a previous non-core node (including an outlier or a newly added node) in the network becomes a core node. The evolutionary event induced due to this particular role change of a node is detected by HOCTracker on checking the following conditions.

If the new core-node  $p$  has mutual-core relations with nodes (visited) that have the same primary community  $C$ , then  $p$  also forms a primary core of community  $C$  by appending this community label to itself and to the nodes in its local  $\varepsilon$ -neighborhood. This simply results in the *expansion* of community  $C$ .

For the new core node  $p$ , if there exist a set of visited core-nodes in the local  $\varepsilon_p$ -neighborhood of  $p$  with which the node  $p$  has mutual-core relations and these core-nodes are primary-core nodes of different communities, then  $p$  causes the primary communities of these core nodes to *merge* into a single community. This is because  $p$  causes the MCMSs of these communities to join and form a single MCMS for the new merged community. The merged community also forms the primary community of the new core node  $p$  and nodes in its local neighborhood are also added to the merged community.

In some other cases, a new-core node may only cause two communities to overlap and not merge. For example, in figure 1a an outlier  $p$  at time  $t$  qualifies as a core node at time  $t + 1$  and establishes mutual-cores relationship(s) with the primary-core node(s) of community  $C$ , thus also forming a primary-core of community  $C$ . Node  $p$  causes the outlier  $q$  to join the community  $C$  at time  $t + 1$  as it belongs to the local neighborhood of  $p$  at time  $t + 1$ . Node  $p$  also includes a primary-core  $r$  of community  $D$  in its local neighborhood but does not show a mutual-core relation with it. This results the two communities  $C$  and  $D$  to overlap with node  $r$  at time  $t + 1$ . Similar, results are demonstrated by figure 1b, however in this case an existing non-core member  $p$  of community  $C$  at time  $t$  qualifies as a core-node at time  $t + 1$  and causes two communities to overlap.

If the new core node  $p$  has no mutual-core relations, then  $p$  forms a new community and appends the new community label to its local neighborhood and itself. This causes the *birth* of a new community with  $p$  being its only primary core.

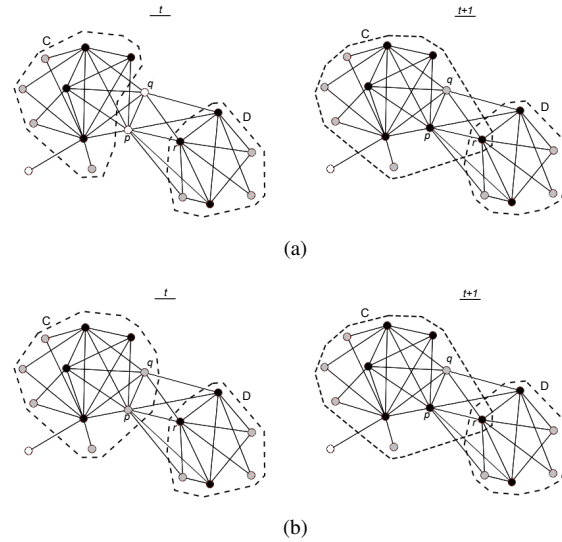


Fig. 1: New core-node causing two communities to overlap

A special case results when a non-core member of a community  $C$  at a previous time qualifies as a core-node at a later stage but its local neighborhood contains nodes that already belong to community  $C$  only and it also shows mutual-cores relation(s) with the primary-cores of community  $C$  only. In this case the new core-node formed does not result in a visible structural change to any community but only changes the role of a node.

*B. A Core Node Becomes a Non-Core*

In this case, an existing core node no longer remains a core node due to some change(s) in the network. This could trigger either a *split* or a *shrink* event in the evolution of a community as follows.

Let  $p$  be a primary core node of a community  $C$  at an earlier time  $t$ , and  $p$  cease to exist as a core-node on the re-computation of its local neighborhood at a later time due to a change in the network. Let  $S$  be the set of core-nodes with which  $p$  had mutual-core relations considering its previous local neighborhood at time  $t$ . We mark the nodes in  $S$  as un-crawled. For a core node  $q \in S$ , let  $B$  be a simple BFS Crawl of nodes starting from  $q$ , wherein all the nodes in the local neighborhood of  $q$  at time  $t + 1$  are appended to the crawl and the mutual-cores of  $q$  are added to a queue. Another core-node then is selected from the front of the queue, its local neighborhood appended to the crawl and its mutual-cores (if any) added to the queue before it is removed from the queue. The crawl  $B$  is completed by repeating this process for each core-node in the queue until the queue is empty. It should however be noted that during a crawl, new local  $\varepsilon$ -neighborhoods of nodes being explored are not re-computed but the existing neighborhoods are used.

If a crawl  $B$  does not include all the core nodes in  $S$ , then the nodes in  $B$  form a new community, i.e., the original community  $C$  is *split* as the lost core property of  $p$  causes a cut in the MCMS of  $C$ . New community labels



are appended to the nodes in  $B$  to represent a new split part of community  $C$  and the community label  $C$  for any nodes in this crawl  $B$  are removed. A new crawl is started in a similar fashion for each remaining un-crawled core nodes (selected at random) in  $S$  until no further split of community  $C$  is possible, i.e., no node in  $S$  remains un-crawled after a crawl. For all the nodes in the last crawl a new community label is appended and the old community label  $C$  of the original split community removed from any nodes possessing it. At the end, if node  $p$  and/or any node, which belonged to the local neighborhood of  $p$  at time  $t$ , are still labeled with community label  $C$ , it means that they do not belong to any intermediate split of community  $C$ . In this case, the community label  $C$  for these nodes is simply removed.

However, if  $B$  includes all the core nodes in  $S$ , then  $p$  is simply removed from being a primary core of community  $C$ . Moreover, if  $p$  and/or any other node that belonged to the earlier local neighborhood of  $p$  are not in the crawl  $B$ , then they are removed with the community label of  $C$ , causing  $C$  to *shrink*. Similarly, a lost-core can cause a community to

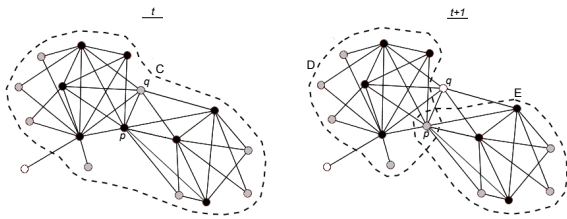


Fig. 2: Lost core-node  $p$  causing a community to split into two overlapping communities

split into overlapping communities. For example, in figure 2 a core-node  $p$  at time  $t$  loses its core property at time  $t+1$  and causes the split of community  $C$  into two communities  $D$  and  $E$  leaving out node  $q$  as an outlier. Moreover, in some cases a core-node may lose its core property but its local neighborhood and/or itself may still belong to the local  $\varepsilon$ -neighborhood of some other primary-core node(s) of its earlier primary-community causing no change to the community except a lost core-property. In this case the lost core-property of node  $p$  does not cause a visible structural change to the community  $C$ . It is also worth to note that in case a lost core node  $p$  was the only primary core-node of a community  $C$ , then it causes the *death* of community  $C$  as no representative primary-core node for community  $C$  remains.

### C. A Core Node Gains nodes (or mutual-core relation(s)) and/or Loses Nodes but Remains a Core

Due to dynamic nature of social networks, changes in them may cause a core node to gain or lose nodes(or mutual-core relations) or both but still hold the core-node property. In this case, the addition or removal of nodes from the local neighborhoods of core-nodes are handled as follows.

1) *Gain*: On re-computing the local  $\varepsilon_p$ -neighborhood, a core-node  $p$  can gain a set  $G$  of nodes to its local neighborhood and/or establish new mutual-core relation with a set  $M$  of existing node(s) in its neighborhood, and still hold the core property. In order to handle these cases, the following steps need to be followed for a sustaining core-node.

- Firstly, we determine the set of visited mutual-core nodes  $G \subseteq N_p^{t+1}$  (wherein  $N_p^{t+1}$  represents the set of nodes in the local  $\varepsilon$ -neighborhood of node  $p$  at time  $t+1$ ) which have a different primary-community than that of  $p$  (if any). In this case, for each node  $o \in G$  the primary-communities of  $o$  and  $p$  are *merged* to form a new community (by replacing the merging community labels of the involved nodes with a new community label for the merged community) and this new merged community now represents the primary-community for both  $p$  and  $o$ .
- Secondly, for any nodes in the set  $N_p^{t+1} - G$  the primary-community label of  $p$  is appended to their respective community lists if they do not contain it resulting in *expansion* of the primary-community of  $p$ . For example, figure 3 shows a primary core-node  $q$  of community  $D$  gain a node  $p$  at time  $t+1$  which is already a member of a different community  $C$ . However, in case of figure 3a the gained node is a non-core while in case of figure 3b the gained node  $p$  is a primary core-node of community  $C$  but does not show a mutual-core relation with  $q$ . In both the cases the two distinct communities  $C$  and  $D$ , at time  $t$ , form two overlapping communities, at time  $t+1$ , which overlap at the common node  $p$ .

It is notable that if a primary-core node  $x$  of a community  $C$  gains a new node  $y$  in its local neighborhood but  $y$  is already a member of the primary community of  $x$ , then no visible structural change occurs to the community  $C$ .

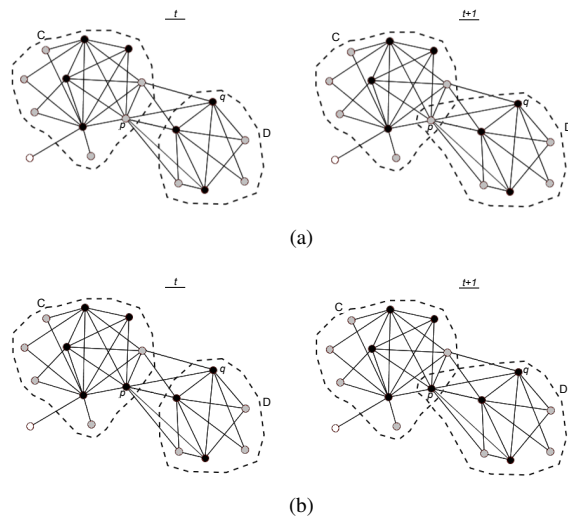


Fig. 3: A core-node gaining other core nodes causing merge of two communities



2) *Loss*: On re-computing the local  $\varepsilon_p$ -neighborhood, if a primary core-node  $p$  of a community  $C$  loses a set of nodes  $S$  and still remains a core-node, then the following procedure is followed.

We mark each node in  $S$  as un-crawled and then for each core-node in  $M \subseteq S$  which had mutual-core relation with  $p$  earlier, we start the crawls in a similar way as discussed in section A-B. If more than one crawls are required to include all the nodes in  $M$ , then each resulting crawl represents a *split* part of the original community and forms a new community. A single crawl including all nodes in  $U$  does not result in a split of the original community and if this crawl includes all the nodes in  $S$ , then no visible structural change is caused to the original community. If any node  $r$  in  $S$  remains which is not included in some crawl(s), then community label of the community whose core-node  $p$  had lost some nodes including  $r$  is removed from node  $r$  causing a *shrinkage* of that community.

## APPENDIX B HEURISTICS FOR PARAMETER $\eta$

It is desirable to find an optimal value for parameter  $\eta$  which reveals the best possible community structure for a given network. It is observed that a good approximation of  $\eta$  is the value at which the resulting community structure has best modularity score [1]. A sensitivity analysis of parameter  $\eta$  at resolution values in the range  $0 < \eta \leq 1$  with step size of 0.1 on two networks with 50,000(synthetic network) and 236(real-world) nodes is provided in figures 4 and 5. Figure 4 gives the size distribution of communities identified at different values of  $\eta$ , wherein x-axis represents community size in a log-scale and y-axis represents the number (in a log-scale) of communities identified by HOCTracker. Figure 5 presents a description of the

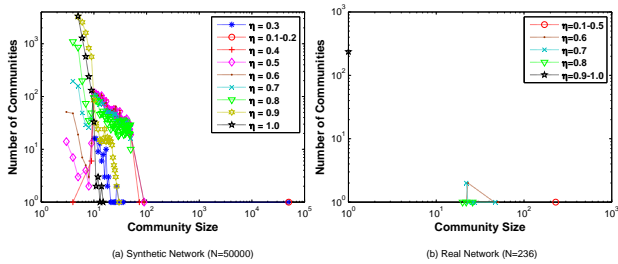


Fig. 4: Size-Density distribution of communities identified by the proposed framework on two networks at different values of  $\eta$

quality of identified communities based on the modularity score  $Q$ . Although, the values of  $Q$  can range between  $[-1$  to  $1]$ , we only consider positive values of  $Q$  and mark negative values (if any) as 0. In addition to this, fractions of total nodes marked as outliers detected at different values of  $\eta$  are also given. Based on the results presented in figure 4 it can be argued that HOCTracker identifies less number of large-sized communities for smaller values of  $\eta$  but a large number of small-sized communities for larger values of  $\eta$ . Figure 5 illustrates that HOCTracker yields low quality

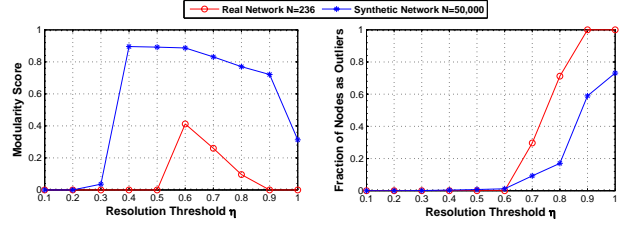


Fig. 5: Modularity score of the community structure and fraction of outliers detected at different values of  $\eta$

communities (in terms of modularity score) at very small and very large values of  $\eta$  and good quality communities in between. However, it is difficult to directly define a generic value for  $\eta$  to detect best community structure in all networks. Although, a trend that can be observed from figure 5 is that for increasing values of  $\eta$  between 0 and 1, modularity of identified community structure first increases very sharply and then decreases slowly. Moreover, higher values of  $\eta$  also result in a large fraction of outliers from underlying networks as it makes the criteria to qualify as core node difficult.

In order to reduce the domain of  $\eta$  for HOCTracker, community structure is identified at each value between the interval  $\eta_{min}$  to  $\eta_{max}$  (starting from  $\eta_{min}$ ) with increment *step* until the next community structure does not result gain or no-change in modularity score  $Q$  [1]. It means starting from a coarser community structure (large and less-dense) HOCTracker moves toward a finer community structure (smaller and more-dense) with a constant step-size of *step*. Now the problem of deciding the value of  $\eta$  involves identifying the values for  $\eta_{min}$ ,  $\eta_{max}$  and *step* for a given network. To define such a domain of  $\eta$  for a given network, HOCTracker considers the *topological-overlap*  $\varphi$  (equation 1) between a pair  $(i, j)$  of nodes connected with an edge.

$$\varphi = \frac{|N_i \cap N_j|}{\min(|N_i|, |N_j|)}, \quad (1)$$

In equation 1,  $N_i$  and  $N_j$  represents sets of nodes to which nodes  $i$  and  $j$  have out-links respectively. The mean ( $\varphi_{mean}$ ) and standard\_deviation ( $\varphi_{std}$ ) of  $\varphi$  are taken over all pairs of nodes between which there exists an edge in the underlying network (rounded-up to two decimal places). Now, the values for  $\eta_{min}$  and *step* are taken as follows:

$$\eta_{min} = \begin{cases} \varphi_{mean} & \text{if } \varphi_{mean} > \varphi_{std} \\ \varphi_{mean} + \frac{\varphi_{std}}{2} & \text{otherwise} \end{cases} \quad (2)$$

$$step = \begin{cases} \varphi_{std} & \text{if } \varphi_{mean} > \varphi_{std} \\ \frac{\varphi_{std}}{2} & \text{otherwise} \end{cases} \quad (3)$$

Once value for  $\eta_{min}$  is decided, the value for  $\eta_{max}$  is taken as follows:

$$\eta_{max} = \eta_{min} + \varphi_{std} \quad (4)$$

It should be noted that for a dynamic network, the value of  $\eta$  is estimated for each new state of a network. In the worst case, for any given state of a dynamic network,

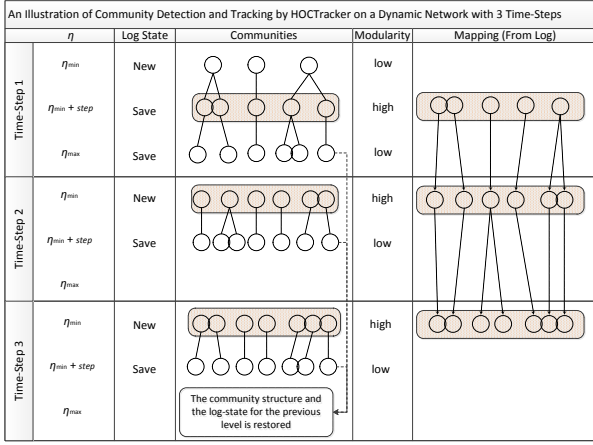


Fig. 6: An illustration of community detection and tracking by HOCTracker on a dynamic network with 3 time-steps

HOCTracker involves using only 2 values for  $\eta$  in case  $\varphi_{mean} > \varphi_{std}$  and 3 values for  $\eta$  otherwise. Before community structure is identified for a new level, community-structure and state of evolution-log for the previous level are saved. If the modularity score for next level is less than previous, the saved state is reloaded to represent communities for current-state of the network and used for mapping evolutionary relations. A simple illustration of community detection and tracking by HOCTracker on a dynamic network involving 3 time-steps is shown in figure 6.

### APPENDIX C HIERARCHICAL DRILLDOWN AND ROLLUP OPERATIONS

As mentioned in the paper, using DrillDown and RollUp operations, the hierarchical mapping between the communities at two consecutive levels can be identified by the same method used for tracking community evolutionary events. It can be seen that in terms of identifying a community hierarchy, the approach followed for DrillDown operation (algorithm 1) is more efficient than the RollUp operation (algorithm 2), as the former involves processing only core nodes of the communities at higher level to identify communities at a lower level. On the other hand, algorithm 2 involves processing all nodes in the network to identify communities at a higher level from communities at a lower level.

### APPENDIX D OVERLAPPING COMMUNITIES AND POST-MERGE

As mentioned in the paper, the proposed community detection method identifies overlapping community structure in a social network. It does so by allowing a node  $q$  to belong to the  $\varepsilon_p$ -neighborhood of a core-node  $p$  irrespective of  $q$ 's previous community assignments in a density-based context as discussed in the paper. Thus a node can belong to multiple communities representing a node where

#### Algorithm 1: DrillDown( $\mathbb{N}, C_{\eta}^L, \eta^+$ )

```

/*  $C_{\eta}^L$  is the community structure of level  $L$ 
   identified at a particular value of  $\eta$  */
/*  $\mathbb{N}$  is the network state for which the
   hierarchy is to be determined */
/*  $\eta^+$  is the new higher value for parameter  $\eta$ 
   at which the communities for level  $L-1$  need
   to be determined */
1 begin
2    $S \leftarrow$  The set of all core nodes in  $C_{\eta}^L$ ;
3    $C_{\eta^+}^{L-1} \leftarrow$  AdaptCS( $\mathbb{N}, C_{\eta}^L, \eta^+, S$ );
4    $G \leftarrow$  Map(Log); /* Log is the transition log
   maintained by AdaptCS */
   /*  $G$  is the bipartite graph which now
   represents the hierarchical relations
   between the community structures  $C_{\eta}^L$  and
    $C_{\eta^+}^{L-1}$  */
5   return  $G$ ;
6 end
    
```

#### Algorithm 2: RollUp( $\mathbb{N}, C_{\eta}^L, \eta^-$ )

```

/*  $C_{\eta}^L$  is the community structure of level  $L$ 
   identified at a particular value of  $\eta$  */
/*  $\mathbb{N}$  is the network state for which the
   hierarchy is to be determined */
/*  $\eta^-$  is the new lower value for parameter  $\eta$ 
   at which the communities for level  $L+1$  need
   to be determined */
1 begin
2    $S \leftarrow$  The set of all nodes in the network state  $\mathbb{N}$ ;
3    $C_{\eta^-}^{L+1} \leftarrow$  AdaptCS( $\mathbb{N}, C_{\eta}^L, \eta^-, S$ );
4    $G \leftarrow$  Map(Log); /* Log is the transition log
   maintained by AdaptCS */
   /*  $G$  is the bipartite graph which now
   represents the hierarchical relations
   between the community structures  $C_{\eta}^L$  and
    $C_{\eta^-}^{L+1}$  */
5   return  $G$ ;
6 end
    
```

multiple communities overlap. It is often possible that two communities overlap in such a way that majority of nodes (more than 50%) of one community (in some cases both the communities) are involved in the overlap between the two communities. In such cases two overlapping communities can be merged to represent a single community.

Following a similar approach, HOCTracker takes  $\eta_{max}$  (discussed in Appendix B) as the overlapping threshold to define the merging criteria as follows. After a community structure is determined by HOCTracker from some state of the underlying network at a particular value of  $\eta$ , HOCTracker merges two overlapping communities if the fraction of nodes, involved in the overlap, for the smaller community is more than or equal to  $\eta_{max}$ . For HOCTracker this process is termed as *post-merge* and is formalized in algorithm 3 and the post-merge step is applied after the main community detection process is completed for a particular value of  $\eta$ . Moreover, it should be noted that post-merge is applied in all the experiments performed in this paper.

---

**Algorithm 3: PostMerge( $C_\eta, \eta_{max}$ )**


---

```

/*  $C_\eta$  is the community structure identified
from the underlying network at a given value
of  $\eta$  and for which the maximum value for  $\eta$  is
taken as  $\eta_{max}$  */
1 begin
2 MergeList is empty;
3 FinalList is empty;
4 foreach pair of overlapping communities ( $C_\eta[i], C_\eta[j]$ ) in  $C_\eta$ 
do
5     if ( $C_\eta[i] \cap C_\eta[j] \geq$ 
6         ( $\eta_{max} \times \min(C_\eta[i].size, C_\eta[j].size)$ )) then
7         add  $i$  to the set MergeList[j];
8         add  $j$  to the set MergeList[i];
9     end
10 end
11  $k \leftarrow 1$ ;
12  $l \leftarrow 1$ ;
13 while  $l \leq C_\eta.size$  do
14     if MergeList[l] is not empty then
15         push(l);
16         while Stack is not empty do
17              $m \leftarrow pop()$ ;
18             foreach  $n$  in MergeList[m] do
19                 if not  $C_\eta[n].merged$  then
20                     add  $n$  to FinalList[k];
21                      $C_\eta[n].merged \leftarrow true$ ;
22                     push(n);
23                 end
24             end
25         end
26          $k \leftarrow k + 1$ ;
27     end
28      $l \leftarrow l + 1$ ;
29 end
/* At the end of the process, each entry in
the array, FinalList, represents the set of
IDs (indices) of communities in the actual
community structure  $C_\eta$  which are merged to
represent a single communities */

```

---

## APPENDIX E

## ADDITIONAL EXPERIMENTAL RESULTS

This section presents the experimental evaluation of HOCTracker in comparison some state-of-the-art community detection methods on real-world and synthetic social networks presented in the paper. These methods include the overlapping community detection method MOSES which has also been used for dynamic community tracking in [2]. Another method used here is the dynamic community detection methods AFOCS which also adapts an initially found community structure to the changes in the underlying network. However, this method does not define an explicit way to track the evolution of the communities across time-steps so as to define the evolutionary events like birth, merge, growth and so on, and only gives the final community structure at the specified time-step. The third method used here SHRINK involves a density-based approach (similar to the proposed method here) to identify disjoint communities from networks along with hubs and outliers. The method SHRINK is included here to show the variation in the identified community structure (if any) by using two related approaches (i.e. SHRINK and HOCTracker). The remaining methods used here include

COPRA, SLPA, CFinder, OSLOM and iLCD.

**A. Results on Real-World Static Networks**

We have used six well-known static real-world network benchmarks to evaluate the performance of HOCTracker and compare it with other state-of-the-art methods based on the NMI (generalized to overlapping communities) [3] and Modularity [1] scores obtained as shown in figure 7. The six networks include the Zachary’s network [4] which is a weighted interaction network between 34 members of a Karate club that split into two communities, the NCAA College football network [5] which is a social network consisting of 115 college football teams divided into eleven conferences and five independent teams, the Dolphin network [6] which is an un-directed and un-weighted social network of frequent associations between 62 Dolphins consisting of two communities, the US political books network is a network of 105 books about US politics<sup>1</sup> sold online by Amazon, and a collaboration network [7] of 241 physicians. For all six real-world networks, the ground truth community structures are known and are used to calculate the performance scores.

As can be seen from figure 7, for each of the six real world networks used, HOCTracker (HT) performs better than all the other methods in question on both the NMI and modularity scores. This is because HOCTracker gets NMI score closer to 1 for all the networks used here and also provides a better Modularity score (closer to 1) for each of the networks.

**B. Results on Dynamic Networks**

It is difficult to evaluate the performance of a dynamic community detection method. This is mainly due to the non-availability of proper benchmarks on which the evolutionary characteristics of communities (birth, death, split and so on) could be explained. Even though, there exist some datasets which involve the dynamics of discussion topics, interests and so on, their underlying interaction networks rarely show characteristics of true social networks (like existence of groups).

Alternately, some synthetic network generation methods (with embedded communities) like the LFR-benchmarks [8] have been proposed. However, as indicated by some researchers like [9] these synthetic network generation methods are often seen to make un-realistic assumptions to generate the networks and hence may not resemble the real-world social networks in the true sense. Moreover, in case of the synthetic networks with embedded overlapping communities, generated by the [8], we have observed that the overlapping nodes (nodes at which communities overlap) indicated by the generator actually qualify as hubs (i.e., nodes which do not belong to any community but connect multiple communities). These hubs are not true overlapping nodes considering the multiple community

<sup>1</sup><http://www.orgnet.com/>

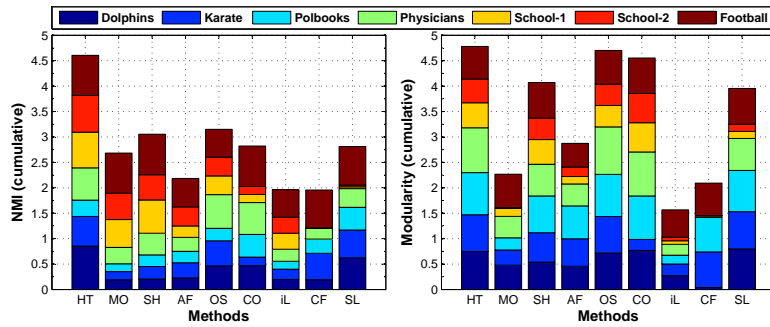


Fig. 7: Experimental results on static real-world benchmarks

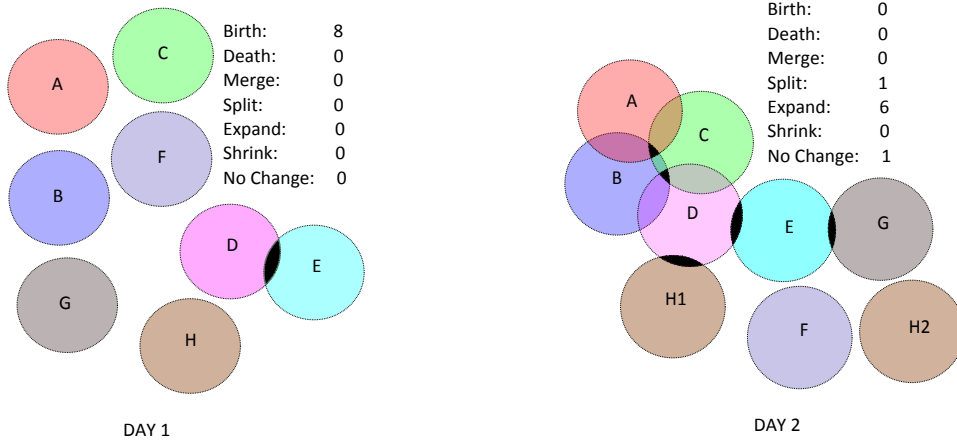


Fig. 8: The evolution of communities over two days in the primary school face-to-face proximity network

membership shown by true overlapping nodes in real-world social networks. In this regard this paper presents a simple analysis of the community structure identified by HOCTracker on some real-world and synthetic networks (where ground truth is known).

1) *Primary School Proximity Network*: In section E-A of this supplement, we presented results on a primary school network which consisted of two networks representing the proximity of individuals over the period of two days. Figure 7 showed that HOCTracker performed better than the other methods in question considering the two networks separately. However, this section considers the dynamic aspect of the network, i.e., two days representing two snap-shots of the network where the *day 1* network is the whole network at the end of first day and *day 2* network represents the changes/additions made to this network in the second day. In this setting, HOCTracker first identifies the preliminary community structure from the *day 1* network and then adapts this community structure to the changes/additions from the *day 2* network. Figure 8 gives a brief representation of the evolutionary relations between the communities identified by HOCTracker on this dynamic network for the two snap-shots.

For the *day 1* network, HOCTracker identifies the birth of 8 communities represented by the circles in figure 8 and labeled from *A* – *H* wherein communities *D* and *E* overlap (represented by black color). After adding the

changes/additions from the *day 2* network to the underlying network, HOCTracker adapts the communities for *day 2* network as shown in figure 8 resulting in 9 communities. Here, community *H* from *day 1* is seen to split into two communities *H1* and *H2* on *day 2* and community *F* remains un-changed. The remaining 6 communities from *day 1* are seen to gain some nodes from their neighboring communities on *day 2* and hence causing their expansion. This expansion also causes some communities to overlap on *day 2*, represented by black colored regions in figure 8. The ground-truth for the primary school network contains 10 communities. The community structure identified by HOCTracker is seen to come more close to the ground truth after adding the information from *day 2* to the information from *day 1* in terms of the number of communities as well as the quality of the communities based on modularity and NMI scores.

2) *Synthetic Dynamic Network*: The LFR-Benchmark generator [8] can be used to generate synthetic power-law networks with embedded distinct or overlapping communities. However, this generator does not consider the dynamic or longitudinal characteristics of evolving networks. On the other hand, Greene et al. [2] extend the LFR-generator to include the dynamic behavior of evolving networks by process of rewiring the edges after networks are generated by LFR-generator. The rewiring involves supervised changes in the community memberships of some nodes thus induc-



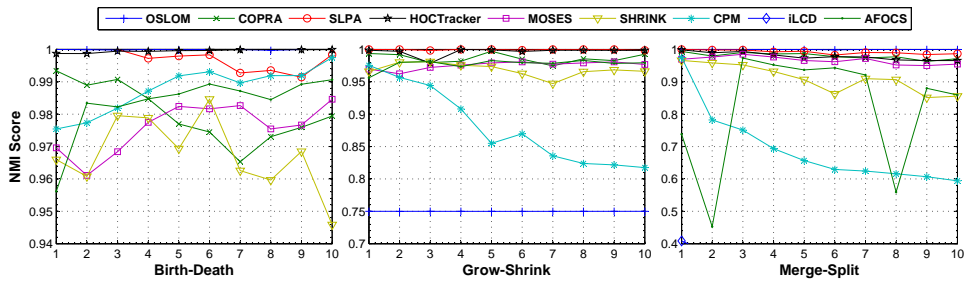


Fig. 9: The NMI scores on synthetic networks with distinct communities

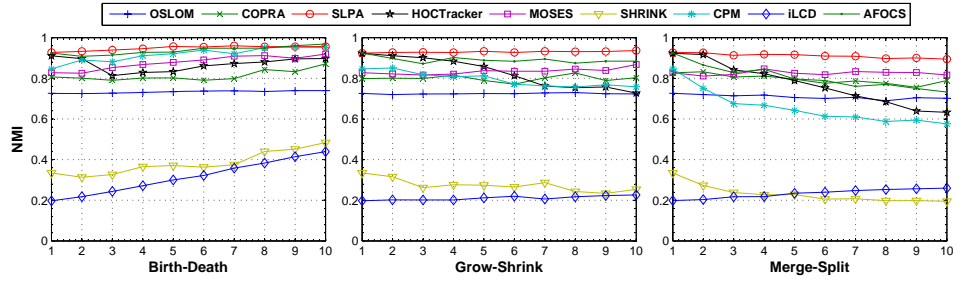


Fig. 10: The NMI scores on synthetic networks with overlapping communities

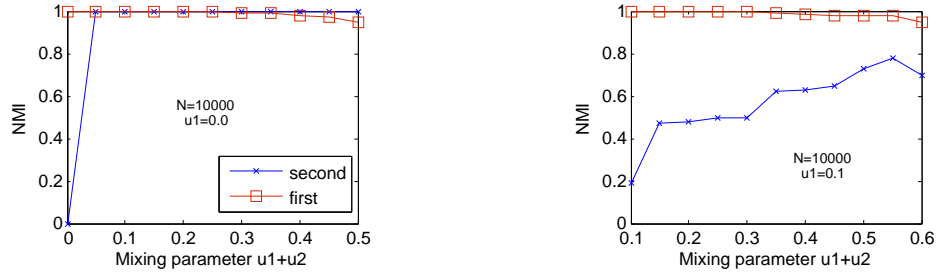


Fig. 11: Results of HOCTracker on un-weighted and un-directed Hierarchical LFR-benchmarks with two levels

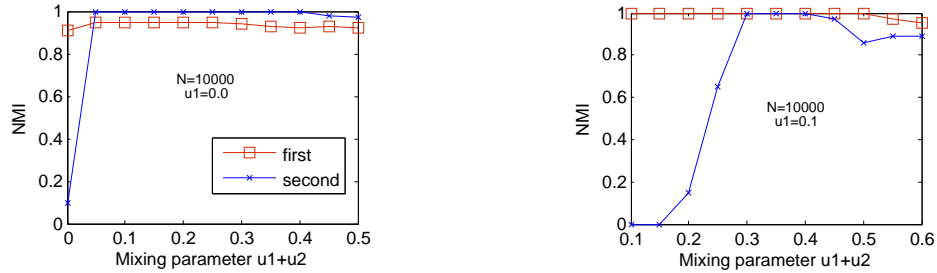


Fig. 12: Results of Infomap on un-weighted and un-directed Hierarchical LFR-benchmarks with two levels

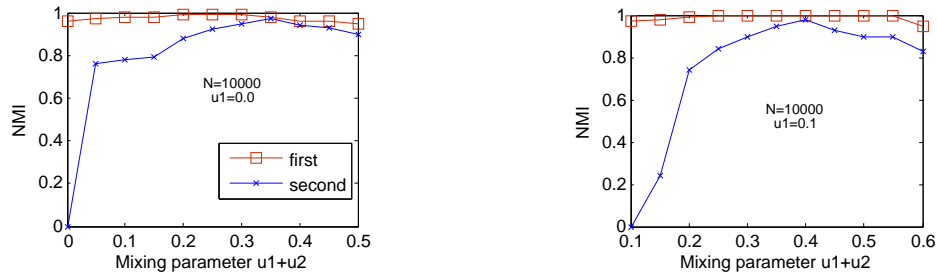


Fig. 13: Results of OSLOM on un-weighted and un-directed Hierarchical LFR-benchmarks with two levels

TABLE I: LFR-Benchmark parameter description and values

Parameter	Description	Value	
		Distinct	Overlapping
$N$	number of nodes	15000	15000
$k$	average degree	20	20
$k_{max}$	max degree	40	40
$C_{min}$	minimum community size	20	20
$C_{max}$	maximum community size	60	60
$\tau_1$	degree exponent	-2	-2
$\tau_2$	community exponent	-1	-1
$\mu$	mixing parameter	0.2	0.3
$O_n$	overlapping nodes	0	10% of $N$
$O_m$	communities per node	1	3

ing a particular number of specified evolutionary events into communities within networks. Using their method, we generate two sets of dynamic networks containing distinct and overlapping communities respectively. The basic parameter settings used to generate the LFR-benchmarks are shown in table I. In order to introduce community-based evolutionary events in the networks, the approach of Greene et al. [2] has been used to generate 10 snap-shots per event-pair for each set as follows.

*Birth and Death-* For each new time-step, it is specified that 40 new communities be introduced (birth) and 40 old communities be removed (death) from the network state of the previous snap-shot. The dynamic graph generator of Greene et al. [2] accordingly introduces 40 birth and death events each in the networks for every snap-shot.

*Merge and Split-* For each new time-step, it is specified that 40 merge events and 40 split events be introduced in each new network state. However, it is seen that the dynamic graph generator of Greene et al. [2] introduces an arbitrary number of these events for each new network state.

*Shrinkage and Growth-* For generating the networks involving shrinkage and growth events, it is specified that 40 shrinkage events and 40 growth events be introduced in each new network state. These events are accordingly introduced in the benchmarks by changing the memberships such that communities lose or gain 25% of their nodes respectively.

In order to provide the performance comparison of the each method in question, we give the NMI score of its identified community structure on various networks. Figure 9 presents the NMI scores of the community structure identified by various methods on synthetic dynamic networks (of 10 snapshots each) containing distinct communities with embedded Birth/Death, Shrink/Growth and Merge/Split. In case of HOCTRACKER, a preliminary community structure is identified from the first snapshot for each dynamic network. Then for each consecutive network snapshot, the changes are added to the underlying network and the community structure is adapted as described in the paper. For AFOCS, we observe that it does not perform correctly (in a dynamic context) on the current networks as they involve removal of some old edges to generate a new network state. In this case we generate results for AFOCS

on each snapshot of a dynamic network separately (i.e., considering each snapshot as a static network). This is also the approach followed to generate the results for rest of the methods used here.

Similarly figure 10 presents the NMI scores of various methods on synthetic dynamic networks containing overlapping communities with embedded Birth/Death, Shrink/Growth and Merge/Split. As mentioned in section E-B, the overlapping nodes in the synthetic LFR-benchmarks actually qualify as hubs (at-least in a density-based context). In this regard, for SHRINK and HOCTRACKER, we enable hub detection in case of networks with overlapping communities and consider the identified hubs as overlapping nodes (besides the actual overlapping nodes detected by HOCTRACKER) to generate the scores.

Figure 9 indicates that for the case of synthetic dynamic networks with distinct communities, HOCTRACKER and SLPA perform comparable to each other and better than the other methods as their NMI scores are mostly close to 1 for all the three case. Method OSLOM also performs well but it scores poorly for the case of Growth and Shrinkage of communities. On the other hand, for synthetic networks with overlapping communities, HOCTRACKER is seen not to perform so well when compared to SLPA and AFOCS. However, given that the overlapping nodes in these synthetic networks do not qualify the same in the proposed approach, the results of HOCTRACKER are still comparable and better than SHRINK which is a related density-based method, and some other state-of-the-art community detection methods.

3) *Results on Hierarchical LFR-benchmarks:* In this section we compare the results of HOCTRACKER with the results of two state-of-the-art methods for detecting hierarchical communities viz Infomap [10] and OSLOM [11] on un-weighted and un-directed hierarchical LFR-benchmarks with two levels of community structures as reported in [11]. In order to generate a two level hierarchical structure, two topology mixing parameters are required:  $\mu_1$ , the fraction of neighbors of each node belonging to different macro-communities;  $\mu_2$ , the fraction of neighbors of each node belonging to the same macro-community but to different micro-communities. Two sets of hierarchical benchmarks have been generated, once by setting  $\mu_1 = 0.0$  and then varying  $\mu_2$  between 0.0 – 0.5, and second by setting  $\mu_1 = 0.1$  and varying  $\mu_2$  between 0.0 – 0.5. Figures 11, 12 and 13 show the plots for NMI scores of the hierarchical community structures identified by HOCTRACKER, Infomap and OSLOM respectively. The x-axis shows the sum  $\mu_1 + \mu_2$ , representing the fraction of neighbors of a vertex not belonging to its micro-community. The plots show how well a method identifies both the *first* and the *second* level of communities from the benchmarks. In order to identify the first level community structure by HOCTRACKER, the global percentage parameter  $\eta$  is varied between 45% – 50% for both  $\mu_1 = 0.0$  and  $\mu_1 = 0.1$  benchmarks to get the best results. To identify the second level community structure,  $\eta = 13%$  for  $\mu_1 = 0.0$

benchmarks and  $\eta$  is varied between 30% – 35% for  $\mu_1 = 0.1$  benchmarks to get the best results. We find that for  $\mu_1 = 0.0$  and  $\mu_2$  between 0.0 – 0.5, HOCTracker performs better than both Infomap and OSLOM as most of the times its NMI score for both the levels in the benchmarks is 1 representing a perfect match with the underlying hierarchical community structure. However, for  $\mu_1 = 0.1$  and  $\mu_2$  between 0.0 – 0.5 HOCTracker finds it more difficult than Infomap and OSLOM to identify the second level communities. Moreover, the performance of all the methods starts to decrease as  $\mu_2 = 0.5$ .

## REFERENCES

- [1] M. E. Newman and M. Girvan, “Finding and evaluating community structure in networks,” *Physical Review E*, vol. 69, 2004.
- [2] D. Greene, D. Doyle, and P. Cunningham, “Tracking the evolution of communities in dynamic social networks,” in *Proceedings of the 2010 International Conference on Advances in Social Networks Analysis and Mining*, ser. ASONAM '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 176–183.
- [3] A. Lancichinetti, S. Fortunato, and J. Kertész, “Detecting the overlapping and hierarchical community structure in complex networks,” *New Journal of Physics*, vol. 11, p. 033015, 2009.
- [4] W. W. Zachary, “An information flow model for conflict and fission in small groups,” *Journal of Anthropological Research*, vol. 33, pp. 452–473, 1977.
- [5] M. Girvan and M. E. Newman, “Community structure in social and biological networks,” in *Proceedings of the National Academy of Sciences*, vol. 99, 2002, pp. 7821–7826.
- [6] D. Lusseau, K. Schneider, O. J. Boisseau, P. Haase, E. Sloaten, and S. M. Dawson, “The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations,” *Behavioral Ecology and Sociobiology*, vol. 54, pp. 396–405, 2003.
- [7] R. S. Burt, “Social contagion and innovation: Cohesion versus structural equivalence,” *American Journal of Sociology*, vol. 92, no. 6, pp. 1287–1335, 1987.
- [8] A. Lancichinetti and S. Fortunato, “Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities,” *Physical Review E*, vol. 80, p. 016118, 2009.
- [9] R. Cazabet, F. Amblard, and C. Hanachi, “Detection of overlapping communities in dynamical social networks,” in *Social Computing (SocialCom), 2010 IEEE Second International Conference on*, aug. 2010, pp. 309–314.
- [10] M. Rosvall and C. Bergstrom, “Multilevel compression of random walks on networks reveals hierarchical organization in large integrated systems,” *PLOS ONE*, vol. 6, p. e18209, 2011.
- [11] A. Lancichinetti, F. Radicchi, J. J. Ramasco, and S. Fortunato, “Finding statistically significant communities in networks,” *PLOS ONE*, vol. 6, no. 5, 2011.