Communities in Streaming Graphs: Small Space Data Structure, Benchmark Data Generation, and Linear Algorithm

Shubham Gupta [1] and Suman Kundu [2]

[1]Indian Institute of Technology Jodhpur
[2]Affiliation not available

October 31, 2023

Abstract

Identifying and preserving community structures in a streaming graph is a very challenging task. However, many applications require the identification of these communities in very limited space and time. In this paper, we design Community Sketch, a small space data structure that efficiently preserves communities. On query, it provides communities in constant time. With the use of community sketch data structure, a linear streaming community detection algorithm is proposed. Experimental results on the large real-world networks show that our algorithm outperforms other state-of-the-art algorithms in terms of quality metrics (NMI, F1-score, and WCC). Further, we propose an algorithm to produce benchmark network, namely, Temporal Community Benchmark Dataset (TCBD) which contains both true community labels and temporal information of edges. These synthetic networks are used to validate the proposed algorithm

Communities in Streaming Graphs: Small Space Data Structure, Benchmark Data Generation, and Linear Algorithm

Shubham Gupta, Suman Kundu, *Member, IEEE*

***Abstract*—Identifying and preserving community structures in a streaming graph is a very challenging task. However, many applications require the identification of these communities in very limited space and time. In this paper, we design Community Sketch, a small space data structure that efficiently preserves communities. On query, it provides communities in constant time. With the use of community sketch data structure, a linear streaming community detection algorithm is proposed. Experimental results on the large real-world networks show that our algorithm outperforms other state-of-the-art algorithms in terms of quality metrics (NMI, F1-score, and WCC). Further, we propose an algorithm to produce benchmark network, namely, Temporal Community Benchmark Dataset (TCBD) which contains both true community labels and temporal information of edges. These synthetic networks are used to validate the proposed algorithm.**

***Index Terms*—Streaming community detection, graph stream, community sketch, benchmark dataset, TCBD.**

I. Introduction

Massive data is being generated by different online services and this data often contains relationships within them. As these data are rapidly generated and transferred through the internet, without secondary storage being used, it is referred to as streaming data. One of the challenges is to store them, considering the volume and velocity of the data being generated. While storing is one part of the challenges, another part is processing the data really fast and answering queries about it. If the streaming data is a graph edge, then the stream is called a graph stream. In the case of graph streams, the whole graph can only be constructed if the full stream is stored in the memory. The graph stream is relevant for the study of 'Social Network Analysis', which deals with the problems related to the collection of relationships and is often used to explain complex systems [1]. Social networks are mathematically modeled using graphs where nodes are entities and edges are relationships among the entities. Community detection is a widely studied problem in the area of social networks where a community is a group of nodes that are densely connected to each other and sparsely connected to the rest of the network. While a community detection algorithm identifies communities from a graph, a streaming community detection algorithm identifies communities from the stream of edges (Illustration in Figure 1). Further, in the case of the streaming community detection problems, the stream is considered infinite in length, and a query to get the community structure will return the community structure seen so far. A formal definition of streaming community detection is provided in Section III.

Many algorithms since 2002 [2] have been developed for detecting communities from graphs but only a few algorithms are developed for streaming version of the problem. Newman and Girvan [3] proposed a community detection algorithm based on modularity which measures the strength of the community structure. This was modified by Newman [4] in 2006 to use eigenvectors for a particular characteristic matrix, known as the modularity matrix, for partitioning graphs into communities. Other community detection algorithms were later proposed, including random walk [5, 6], spectral clustering [7, 8, 9], and statistical-inference [10] techniques. Recently, the performance of the modularity function was further improved with fuzzy maximization [11] and correlation clustering [12]. However, these algorithms require storing the entire graph in the form of an adjacency matrix or list, making them impractical for larger graphs. In addition, when used with streaming graphs, these methods require recalculating the community structure from scratch on each query. To address these issues, Hollocou et al. [13] proposed a streaming community detection algorithm that identifies communities in linear time but may not create a good partition of the graph in terms of quality metrics. Another methodology in the streaming setting, seed set expansion algorithms [14, 15] better partition the graph using true communities of seed nodes but labeling these nodes in graph streams is challenging. Some community detection algorithms fail to identify small clusters or create well-structured communities for non-uniform community sizes.

In this work, we have designed a small space community sketch data structure which is a combination of a forest and a sparse triangular matrix. The forest is made of a group of disjoint trees, where each tree represents a community of a set of nodes, and a sparse triangular matrix stores the properties of the community structure. Proposed community sketch data structure supports four operations viz 'community', 'make-sketch', 'update-sketch', and 'merge-community'. It also provides constant query operation to identify the communities at any given point in time. Using this data structure, we have developed a linear time community detection algorithm for the streaming graph where each edge comes in a sequence and is checked if it is creating a community or not. It strictly processes each edge at once to make the computation time

S. Gupta and S. Kundu is with Department of Computer Science and Engineering, Indian Institute of Technology Jodhpur, India - 342030 .
E-mail: gupta.37@iitj.ac.in, suman@iitj.ac.in
Code: https://github.com/vigilante007/Streming-Community-Detection

linear in nature. Two communities are merged based on the edge density within (inner) a community and with (outside) other communities. Extensive experiments are performed on real-world datasets available at SNAP [16]. Our proposed algorithm performs better or comparable than state-of-the-art algorithms in terms of WCC, F1-score and NMI (Normalized Mutual Information) while taking less time and space. However, we observed that publicly available datasets with ground truth communities don't have any temporal information, and datasets with temporal information don't have ground truth communities. This restricts to check the effectiveness of streaming community detection algorithm. In order to get around this limitation, We introduce the Temporal Community Benchmark Dataset (TCBD), which generates benchmark data with ground truth communities and temporal information. We use the LFR benchmark algorithm [17] to generate an initial graph, then iteratively add a new set of edges while maintaining community labels to create a temporal large graph. We provide a detailed comparative analysis of these generated graphs with LFR benchmark graphs. In particular, we sum up contributions of this study as follows:

- This research proposes a small space community sketch data structure which gets, creates, updates, and merge communities in an efficient manner.
- This research delivers a linear community detection algorithm for streaming graphs by storing the communities in proposed community sketch data structure. Algorithm returns the communities in a constant query time at any given point in time.
- To generate the ground truth data for streaming community detection algorithm, this research proposes a Temporal Community Benchmark Dataset algorithm which generates the ground truth communities with temporal information.
- Extensive experiments are performed to check the effectiveness on proposed streaming community detection algorithm in terms of quality metrics. Change in modularity and conductance values over time is also derived for the proposed algorithm in the streaming setting.

The paper is organized as follows: Section II reports the related work. Section III defines the problem statement formally. Sections IV and V describe our proposed algorithm and proposed benchmark dataset. Sections VI and VII show the theoretical and experimental analysis respectively; and finally, Section VIII concludes the research outcome.

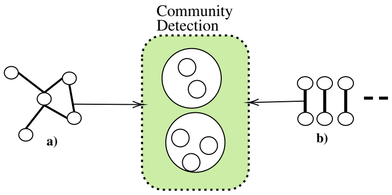

Fig. 1: Community detection in a) graph and b) graph stream.

II. Related Work

In the literature, Community detection techniques can be found in two different settings 1) non-streaming and 2) streaming. Before discussing the literature for these two categories, let us first discuss some of the quality metrics used in many community detection algorithms [18]. One of the popular metrics is modularity [4, 11, 12], which calculates the difference between the number of edges observed in each cluster and those were randomly distributed in the cluster. Another well known metric is conductance [19, 20], the ratio between the relationship of edges outside a community and within a community. Other metrics have been used like weighted community clustering [21], out-degree fraction [22] etc.

A. Non-streaming

Pioneer work on community detection was done by Girvan and Newman 2002 [2]. They successively published different community detection algorithms [3, 4]. The former algorithm iteratively removes the edges from the network based on edge centrality and remaining connected components in the network are referred as communities. In the later algorithms, they defined modularity and optimized it to get the community structure. The Louvian algorithm [23] improves modularity by running each iteration in two phases: first, each node is placed in its own community, and modularity gain is calculated; secondly, a new network is created using the communities discovered in the first phase. Correlation clustering [12] and fast fuzzy modularity maximization [11] have been used to enhance classical community detection algorithms using the modularity function. Random walk-based algorithms [5, 24] involve using a random surfer to identify the neighbourhood and tend to become trapped in the densest section of the graph. Palla et al. [25] proposed Clique Percolation Method which expands cliques to find dense communities. Classical clustering techniques are also used to finding communities. SCAN [9] is one of such algorithms that places the densely connected adjacent nodes in same clusters by using the geometric mean of their degrees to determine structural similarity. Some of the parallel processing based community detection algorithms [26, 27] are also proposed to expedite the execution of tasks in the community detection. Raghavan et al. [28] proposed Label Propagation Algorithm (LPA) which assigns community based on the majority votes by neighbourhood. In recent years, randomness and instability problem in LPA was improved with modularity function [29] and $k-$core model [30] respectively. All of the above methods were shown to be effective, however these are not easily portable for the streaming graphs.

B. Streaming

Only a few community detection algorithms are proposed for the streaming graphs. Hollocou et al. [13] presented one of the very first work in streaming community detection. The algorithm is called SCODA, where it only stores degree per node and communities are formed based on the same. In the extension of SCODA, Sabour and Moeini [31] first found out the maximum clique from the graph and then provide all the maximal clique to the SCODA as input. Further, SCAOD [32] was designed which used the concept of node contribution with the condition of SCODA that the probability of the coming edge is an edge within a community is much greater than that of an edge between communities. On the other hand, some seed set expansion techniques [14, 15, 33] for streaming graphs have been proposed. Liakos et al. [14] developed the COEUS algorithm, which measures the edge quality in relation to the community. In extension to COEUS, DICES [15] was proposed to execute COEUS in a distributed fashion in order to make it computationally light. Further in 2022 [33], authors improved the performance of the COEUS by introducing one additional index. These seed set expansion methods need some kind of guidance or ground truth to identify the communities from the network. Incremental methods are also proposed such as incremental k-core based decomposition algorithm [34] where k-core decomposition is updated in algorithm by finding a small subgraph. In order to speed up insertion and deletion operations even further, Sarıyüce et al. [35] updated the incremental algorithms and proposed auxiliary vertex state management strategies. Recently, Wu et al. [36] proposed streaming belief-propagation approach for community detection that work on the limitations of voting algorithms.

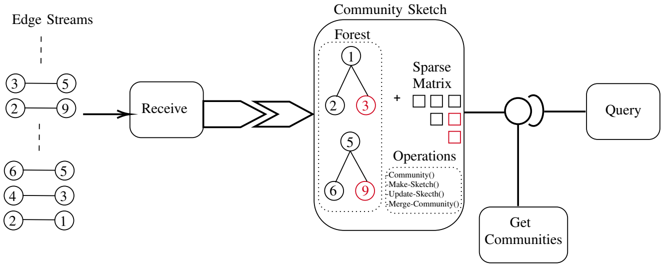

Fig. 2: Workflow of the proposed algorithm.

III. Problem Statement

Let S be a graph stream defined as $S = \langle e_{t_1}, e_{t_2}, e_{t_3} \ldots, \rangle$, where an edge $e_{t_i} = (u_{t_i}, v_{t_i})$ is a unique unordered pair. Also, let $G_t(V_t, E_t)$ is the undirected unweighted graph seen till time t. Where $V_t = \{v | v \in \bigcup_{t_i < t} \eta(e_{t_i})\}$ and $E_t = \{e_{t_i} | t_i < t\}$. The function $\eta(e_{t_i} = (u_{t_i}, v_{t_i}))$ returns a set $\{u_{t_i}, v_{t_i}\}$ for a given edge e_{t_i}. Formally, a streaming community detection problem on graph stream S is to mine a set of communities $C_t = c_1, c_2, \ldots, c_k$ from graph $G_t(V_t, E_t)$ for any t.

In the literature, there is no universal definition available for defining what communities are; yet, most of the algorithms rely on the principle that a community is represented by a set of nodes of the graph that are closely connected to each other and have loosely connected with the rest of the network [3]. There are many quality metrics available in the field of community detection, but conductance and modularity are most commonly used to check the quality of the communities. Formally, conductance $\varphi(C)$ of a community C and modularity Q of graph G are defined as:

$$\varphi(C) = \frac{|e(C, \bar{C})|}{2|e(C, C)| + |e(C, \bar{C})|} \tag{1}$$

where $e(C, C)$ is intra-community edges and $e(C, \bar{C})$ is inter-community edges.

$$Q = \frac{1}{2m} \sum_{ij} \left[A_{ij} - \frac{d_i d_j}{2m} \right] \delta(c_i, c_j) \tag{2}$$

where c_i is group to which vertex i belongs, d_i is degree of node i, A_{ij} is weight of edge between i and j, δ is the Kronecker delta function and m is total number of edges.

IV. Proposed Linear Community Detection Algorithm

We have proposed a linear streaming community detection algorithm to find out communities from the stream in an asynchronous manner. For a streaming community detection algorithm, communities and their properties must be kept in some kind of data structure that can provide communities in constant query time. We have developed a space-efficient data structure called Community Sketch to store communities and their properties, allowing for constant query time. Our proposed streaming community detection algorithm uses the Community Sketch to store communities. The algorithm processes incoming edges one by one, creating a new community based on each edge's relationship and updating the Community Sketch accordingly. The merging of communities occurs based on the inner and outer density concept. Communities can be found asynchronously when a query is made. The workflow of our algorithm is illustrated in Figure 2. A community with a set of nodes will be merged with another community if the outer density is much greater than the inner density. The rationale of this can directly be derived from the definition of the community which says a community have larger number of links inside than inter community. Hence, a merge can only happen when the density of edges increase on merge. Mathematically, inner density (ρ_{in}) and outer density (ρ_{out}) of the communities C_1, C_2 having edges m_1, m_2, nodes n_1, n_2 and cross connecting edges m_{12}, is formulated as below:

$$\rho_{in}(C_1) = \frac{\textit{Total no. of edges present in the community}}{\textit{Total no. of possible edges in the community}} = \frac{2 \times m_1}{n_1.(n_1 - 1)} \tag{3}$$

$$\rho_{out}(C_1, C_2) = \frac{\textit{Total no. of outer edges connecting } C_1 \textit{ to } C_2}{\textit{Total no. of possible outer edges connecting } C_1 \textit{ to } C_2} = \frac{m_{12}}{n_1.n_2} \tag{4}$$

Figure 3 shows the calculation of the inner and outer density of two communities C_1 and C_2 in a network. The following subsections define the proposed community sketch and streaming community detection algorithm in detail along with the operations it supports.

A. Community Sketch

A community sketch is a small space data structure that stores communities for a graph stream S without storing the entire graph G. It comprises a forest and a sparse triangular matrix, where each tree in the forest represents a community of nodes $n_1, n_2, \ldots, n_k$, with the root element serving as the community representative. The sparse triangular matrix stores node and edge count information within (inner) a community and with other (outer) communities. The matrix's diagonal cells have two counters, and the rest have one counter, with the diagonal representing within-community edge and node counts and the others store edge count between communities. The matrix can be fully populated in the worst case, but in practice, there will be many 0's. The community sketch supports four operations: Community(·), Make-Sketch(·), Update-Sketch(·), and Merge-Community(·). The Community(·) function's cost may increase due to continuous tree merging, but path compression helps to reduce its amortized cost. Each of these functions is described below:

Community($node$): It returns a community representative of a community where $node$ belongs. In other words, it returns the root node of a tree where a $node$ lies.

```
def community($node$):
  while $node$ != $node$.parent:
    $node$.parent, $node$ := $node$.parent.parent, $node$.parent
  return $node$
```

Make-Sketch(): It initializes a community sketch $\mathcal{C}_k$ with a forest f and a sparse matrix mat.

```
def makeSketch():
  $\mathcal{C}_k$ = forest $f$, sparseMatrix $mat$
  return $\mathcal{C}_k$
```

Update-Sketch($\eta(e)$): The function updates the Community Sketch with a new edge e arrival. It checks if both nodes belong to existing communities. If they belong to the same community, the edge count within the community is increased. If they belong to different communities, the edge count between communities is increased. If either node is part of an existing community, a new single-node community is created with the remaining node as its representative. If both nodes are not part of any community, a new community with two nodes is created, with the node having the lower value assigned as the community representative and the other node as its child node. The sparse triangular matrix mat is then updated with the new community's node and edge count information.

```
def updateSketch($\eta(e = (node_1, node_2))$):
  $n_1$, $n_2$ = $\mathcal{C}_k$.community($node_1$), $\mathcal{C}_k$.community($node_2$)
  # when both nodes exist
  if ($n_1$ && $n_2$):
    $\mathcal{C}_k$.mat[$n_1$][$n_2$].nedge += 1
  # when both nodes do not exist
  elif (!$n_1$ && !$n_2$):
    $n$ = $n_1$ if $node_1 < node_2$ else $n_2$
    $n_1$, $n_2$ = $\mathcal{C}_k$.$f$.add($node_1$), $\mathcal{C}_k$.$f$.add($node_2$)
    $n_1$.parent, $n_2$.parent = $n$
    $\mathcal{C}_k$.mat[$n$][$n$].nnode, $\mathcal{C}_k$.mat[$n$][$n$].nedge= 2, 1
  # when one of them exists
  else:
    $n$ = $n_1$ if !$n_1$ else $n_2$
    $node$ = $node_1$ if !$node_1$ else $node_2$
    $n$ = $\mathcal{C}_k$.$f$.add($node$)
    $n$.parent = $n$
    $\mathcal{C}_k$.mat[$n$][$n$].nnode, $\mathcal{C}_k$.mat[$n$][$n$].nedge= 1, 0
```

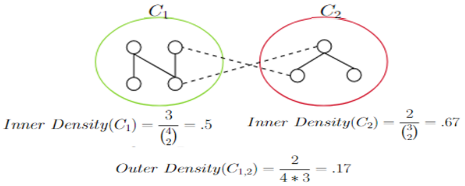
C_1
C_2
$Inner\ Density(C_1) = \frac{3}{\binom{4}{2}} = .5$
$Inner\ Density(C_2) = \frac{2}{\binom{3}{2}} = .67$
$Outer\ Density(C_{1,2}) = \frac{2}{4*3} = .17$

Fig. 3: Inner and Outer Density of two communities.

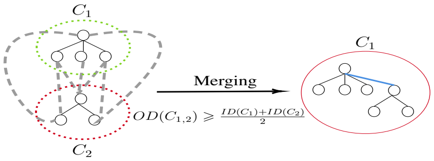

Fig. 4: Merging of two communities.

Merge-Community(P_{n_1}, P_{n_2}): This method merges communities represented by P_{n_1} and P_{n_2} based on the size of the communities, and the smaller community is linked to the larger community representative. The sparse triangular matrix mat is updated after the merge, and entries of the merged community are removed and references are updated to the new community.

```
def mergeCommunity($P_{n_1}$,$P_{n_2}$):
  $P_x$, $P_y$ = $P_{n_1}$, $P_{n_2}$ if $\mathcal{C}_k$.mat[$P_{n_1}$][$P_{n_1}$].nnode
          > $\mathcal{C}_k$.mat[$P_{n_2}$][$P_{n_2}$].nnode else $P_{n_2}$, $P_{n_1}$
  $\mathcal{C}_k$.mat[$P_x$][$P_x$].nnode += $\mathcal{C}_k$.mat[$P_y$][$P_y$].nnode
  $\mathcal{C}_k$.mat[$P_x$][$P_x$].nedge += $\mathcal{C}_k$.mat[$P_y$][$P_y$].nedge
                         + $\mathcal{C}_k$.mat[$P_x$][$P_y$].nedge
  $\mathcal{C}_k$.mat.remove($P_y$)
  $P_y$.parent = $P_x$
  $\mathcal{C}_k$.mat.replaceRef($P_y$) ← $P_x$
```

B. Algorithm

Algorithm 1 is a streaming community detection algorithm for edge streams. It has three operations: initialization, **onReceive**(e), and **onQuery**(). If there is no community sketch, the algorithm initializes one using the **makeSketch()** function. The **onReceive**(e) function updates the data structure using the **Update-Sketch**(·) function. The **Update-Sketch**(·) function identifies the communities of nodes u and v and calculates their outer and inner densities based on edge and node count information from the community sketch data structure. If the outer density is greater than the α (user-defined parameter) times sum of inner density of both communities, the communities can be merged using the **Merge-Community**(·) function, as shown in Figure 4. The **onQuery()** function returns all communities seen in the stream till the query is made. The query function returns all the communities seen in the stream till the query is made. Algorithm 2 can also be used for static graphs by initializing a community sketch and calling **onReceive**(e) for each edge in the input edge list E, then returning all communities found. Essentially, Algorithm 2 randomise the edge list and use it as a stream of edges. Note that because of the randomisation different executions will yield different community structures. Experimentally, we found that this different runs will not have much effect on the quality metrics (e.g., NMI for Amazon varies between 0.163 to 0.170 III) and wherever require we report the best value out of 10 different executions.

C. Time Complexity

The Algorithm 1 processes each edge only once, i.e., time complexity is linear to the size of the stream assuming density calculation and checking merging condition are taking $O(1)$ time. The complexity of the community(·) function depends upon the height of the tallest tree in the forest. However, path compression technique is used while community(·) function is called. The amortized cost of community(·) function per operation is $O(\alpha(n))$ [37], where $\alpha(n)$ is inverse Ackermann function which grows extremely slow and can be considered as constant. Thus, the overall time complexity of **onReceive**(e) function is $O(|E|)$, where E is edge set in the whole stream. Complexity of **onQuery()** function is constant which returns the pointer to all communities. In non-streaming version, Algorithm 2 is running till the processing of edge list then total time complexity is $O(|E|)$.

D. Space Complexity

We are only using two dictionaries of integer, one with size n for keeping the forest as parent pointer. Second dictionary is used to store sparse triangular matrix which will have a size $\frac{|C|^2}{2}$. In worst case $|C|$ can be $|V|-1$ for star graph. However, even with worst case most of the cells in triangular matrix will be 0 and we are not blocking any space for those entries. For the practical scenario, $|C| << n$. The total space complexity of the proposed algorithm is $O(|V|+|C|^2)$. The advantage of the streaming technique is that there is no need to store the entire graph.

Algorithm 1 Streaming Community Detection

```
Input: An edge stream, fraction threshold $\alpha$
  // Intialize Community Sketch
  CommunitySketch $\mathcal{C}_k$
  if $\mathcal{C}_k$ == NULL then
      $\mathcal{C}_k$ = makeSketch()
  end if

  // Receive Function
  def onReceive($e$, $\alpha$):
      $\mathcal{C}_k$.updateSketch($\eta(e = (u, v))$)
      $C_u$=$\mathcal{C}_k$.community($u$)
      $C_v$=$\mathcal{C}_k$.community($v$)
      // calculate outer density of communities $C_u$ and $C_v$
      $\rho_{out}(u, v)$ = $\mathcal{C}_k$.mat.getOuterDensity($C_u$,$C_v$)
      // calculate inner density of community $C_u$
      $\rho_{in}(u)$ = $\mathcal{C}_k$.mat.getInnerDensity($C_u$)
      // calculate inner density of community $C_v$
      $\rho_{in}(v)$ = $\mathcal{C}_k$.mat.getInnerDensity($C_v$)
      if $\rho_{out}(u, v) \geqslant \alpha \times (\rho_{in}(u) + \rho_{in}(v))$ then
          $\mathcal{C}_k$.mergeCommunity($C_u, C_v$)
      end if

  //Query Function
  def onQuery():
      return $\mathcal{C}_k.f$
```

Algorithm 2 Non-Streaming Community Detection

```
Input: List of edges $E$ between nodes $\{1, \ldots, n\}$
Output: A set of communities, $\mathcal{C}_k.f$
  // Intialize Community Sketch
  CommunitySketch $\mathcal{C}_k$ = makeSketch()
  for each $random(e)$ in $|E|$ do
      onReceive($e$)
  end for
  return onQuery()
```

V. Algorithm: Temporal Community Benchmark Dataset (TCBD)

In order to verify the effectiveness of streaming community detection algorithm, ground truth data is necessary. Although, there are ground truth data generators (LFR Benchmark dataset [17]) available for communities in static graph, there is no algorithm available for generating temporal graph with ground truth communities. Further, the ground truth must match the topological changes in the temporal graph for each snapshot.

In the proposed Algorithm 3, a base graph is first generated using the LFR Benchmark algorithm [17]. In subsequent iterations, the algorithm allows the graph to grow with addition of new nodes (say r) with degree generated by power law distribution. We also add new communities in the graph in each iterations. The number of communities added in each iteration is calculated by $N_c = \frac{r}{0.5\times(min(r,k_{\max})+k_{\min})}$. Once N_c is calculated the size of each community is generated with power law distribution similar to the first iteration. After that, nodes are assigned randomly to the new communities with condition that $(1-\mu)$ fraction of links with the same community nodes, while μ fraction of links with nodes in other new communities. The communities thus formed may not contain all the newly added nodes due to two factors (i) degree of the node is larger than the number of nodes in the communities, (ii) r is greater than the sum of community size's. Nodes which are not part of any newly formed communities are referred as "homeless" nodes. Homeless nodes are assigned to communities generated earlier depending on the likelihood determined by preferential attachment. These homeless nodes form internal links in the community and external links with newly formed communities by the rule of mixing parameter μ and play a crucial role of connecting the existing graph with the newly generated graph. The detailed algorithm is presented in Algorithm 3.

Algorithm 3 TCBD Generation

```
Input: List of number of nodes added $r$, average degree $\langle k\rangle$, degree power law exponent $\gamma$, community power law exponent $\beta$, min and max community size $s_{\min}$ and $s_{\max}$, mixing parameter $\mu$
Output: TCBD Graph, $G$
1: Generate $n$ nodes from the list and determine the degree of each by using the three parameters$\langle k\rangle$, $k_{\max}$, and $\gamma$ based on the power law distribution.
2: Generate the communities using $s_{\min}$, $s_{\max}$, and $\beta$ based on power law distribution. This step is having two parts.
  (a) If no initial graph present then the length of community sequence will be sum of all sizes must be equal to $n$.
  (b) If graph exits then The length of community sequence must be equal to $\frac{r}{0.5\times(min(r,k_{\max})+k_{\min})}$ which means sum of all sizes can be equal to $r$ or less than $r$.
3: Get the number of internal and external linkages for each node. Every node has $(1-\mu)$ linkages with other nodes, whereas $\mu$ links are with nodes outside of that community.
4: Create edges for each node that connect it to randomly chosen internal and external nodes in communities, matching the number of internal and external links counted in step 3. This step is having two parts.
  (a) Initially if no graph exists, no nodes are assigned to communities. After that, a community is chosen at random for each node. A new node is added to the community if the number of nearby nodes does not already exceed the community size; otherwise, it remains outside. The "homeless" node is randomly assigned to any community in the following iterations.
  (b) If graph exists, the first homeless nodes are assigned to communities based on the probability decided by preferential attachment. Internal nodes of existing communities are adjusted based on mixing ratio. Next, assign remaining nodes to new community with the same condition as discussed in 4(a).
```

Remark V.1. *Homeless nodes are added to the existing communities with the condition that $(1-\mu)$ links are created within the community, and μ links are created with newly nodes of other communities. However in creation of external links, chances of selecting a homeless node is very less and it will pick the nodes of newly formed communities. This rule may disrupt the mixing ratio of nodes in existing communities that are connected to homeless nodes during this process. As a result, internal links are increased for that particular community, which makes the community structure stronger with denser links. In the proposed algorithm, we reconsider those nodes and create external links whenever required. However, probability of an existing node being selected repeatedly by newly added node is $\frac{1}{|c|}$, where c is the community where a homeless node was added. As already discussed, many homeless nodes have high degree because that is the reason of being homeless. These homeless nodes will anyway be added to a existing large size community. Chances that the remaining homeless nodes select a larger size community is high by likelihood of preferential attachment. It implies majority of homeless nodes will attach to the community having higher cardinality. Hence the probability of node being selected repeatedly is very low i.e. $\alpha = (\frac{1}{|c|})^{\#\text{homeless node}}$. The same is verified experimentally and results are presented in Table I.*

Remark V.2. *During community size generation, it is possible that community size generated by power law distribution will cover the entire n results in no homeless node. In this situation, the graph generated in that snapshot will be disconnected, which is likely to be resolved in the subsequent iterations. Although this situation is rear but can easily be avoided by reserving the amount of homeless nodes by configuration.*

A. Comparative Analysis

We conducted a comparative study of TCBD and LFR Benchmark Dataset by generating graphs of size 8000 with varying mixing parameter μ from 0.1 to 1.0 using both algorithms. For each value of μ, we generated 16 snapshots of the graph by adding 500 nodes in each iteration. We calculated modularity and conductance for the corresponding graphs and found that TCBD outperformed LFR algorithm in terms of modularity and conductance evident in Figures 5 and 6 respectively. In Figure 5, TCBD showed an increase in modularity after adding 500 nodes because generating the very first graph follows the same rule and converged after some time as discussed in Remark V.1, while conductance changes were similar or better than LFR Benchmark algorithm and converged after some time.

TABLE I: Statistics of homeless node added to existing communities.

#Homeless node	Community size	Homeless added (%)	α
13	40	.026	$(1/40)^{13}$
9	52	.018	$(1/52)^{9}$
8	60	.016	$(1/60)^{8}$
7	71	.014	$(1/71)^{7}$
6	64	.012	$(1/64)^{6}$
5	60	.01	$(1/60)^{5}$
4	91	.008	$(1/91)^{4}$
3	101	.006	$(1/101)^{3}$

VI. Theoretical Analysis

Equations 2 and 1 show the modularity of a graph G and the conductance of a community c. In modularity, if both nodes exist in the same community then $\delta(c_i, c_j)$ will be equal to 1. Let us drive the change in Q_t and $\varphi(c_t)$ terms while running the algorithm.

Lemma VI.1. *Given a graph stream S, let Q_t represents the modularity of the graph after time t; this implies t edges are seen so far. For the proposed community detection algorithm, change in modularity Δ_{Q_t} after adding t edges is* $\frac{1}{\sum\limits_i d_i}\left[\sum_{ij} A_{ij}.\delta(c_i,c_j) - \frac{\sum\limits_{ij;i\neq j} d_i.d_j + \sum\limits_i b_i.d_i + B}{\sum\limits_i d_i}\right] - \sum_{i=0}^{t-1}\Delta_{Q_i}$.

Proof: *Let initial modularity and change be Q_0 and Δ_{Q_0}. Therefore, $Q_0 = 0, \Delta_{Q_0} = 0$*

$$\textit{At } t = 1, Q_1 = \frac{1}{2(m+1)}\left[1 - \frac{(d_i+1)(d_j+1)}{2(m+1)}\right]$$

Then change in modularity from Q_0 to Q_1 is,

$$\begin{aligned}\Delta_{Q_1} &= Q_1 - Q_0\\ &= \frac{1}{2(m+1)}\left[1 - \frac{(d_i+1)(d_j+1)}{2(m+1)}\right] - \frac{1}{2m}\left[0 - \frac{d_i d_j}{2m}\right]\\ &= \frac{1}{2(m+1)}\left[1 - \frac{d_i + d_j + d_i d_j + 1}{2(m+1)}\right] - \Delta_{Q_0}\end{aligned}$$

Without the loss of generality, let us assume the incoming edge is connected to existing graph. Calculation for disconnected edges will be more involved.

Now,

$$\begin{aligned}Q_2 = &\frac{1}{2(m+2)}\left[\left[1 - \frac{(d_i+2)(d_j+1)}{2(m+2)}\right] + \left[1 - \frac{(d_i+2)(d_k+1)}{2(m+2)}\right]\right.\\ &\left.+ \left[0 - \frac{(d_j+1)(d_k+1)}{2(m+2)}\right]\right]\end{aligned}$$

And,

$$\begin{aligned}\Delta_{Q_2} &= Q_2 - Q_1\\ &= \frac{1}{2(m+2)}\left[\left[1 - \frac{(d_i+2)(d_j+1)}{2(m+2)}\right] + \left[1 - \frac{(d_i+2)(d_k+1)}{2(m+2)}\right]\right.\\ &\quad\left.+ \left[0 - \frac{(d_j+1)(d_k+1)}{2(m+2)}\right]\right] - \frac{1}{2(m+1)}\left[1 - \frac{(d_i+1)(d_j+1)}{2(m+1)}\right]\\ &= \frac{1}{2(m+2)}\left[\left[1 - \frac{(d_i+2)(d_j+1)}{2(m+2)}\right] + \left[1 - \frac{(d_i+2)(d_k+1)}{2(m+2)}\right]\right.\\ &\quad\left.+ \left[0 - \frac{(d_j+1)(d_k+1)}{2(m+2)}\right]\right] - \Delta_{Q_1} - \Delta_{Q_0}\\ &= \frac{1}{2(m+2)}\left[[0+1+1] - \left[\frac{d_i.d_j + d_i.d_k + d_j.d_k}{2(m+2)}\right]\right.\\ &\quad\left.- \left[\frac{2d_i + 3d_k + 3d_j}{2(m+2)}\right] - \left[\frac{5}{2(m+2)}\right]\right] - \Delta_{Q_1} - \Delta_{Q_0}\end{aligned}$$

$\dots$

$$\begin{aligned}\therefore \Delta_{Q_t} &= \frac{1}{2(m+t)}\left[\sum_{ij} A_{ij}.\delta(c_i,c_j) - \frac{\sum\limits_{ij;i\neq j} d_i.d_j + \sum\limits_i b_i.d_i + B}{\sum\limits_i d_i}\right]\\ &\quad - \left[\Delta_{Q_{t-1}} + \Delta Q_{t-2} + \cdots + \Delta_{Q_0}\right]\\ \Delta_{Q_t} &= \frac{1}{\sum\limits_i d_i}\left[\sum_{ij} A_{ij}.\delta(c_i,c_j) - \frac{\sum\limits_{ij;i\neq j} d_i.d_j + \sum\limits_i b_i.d_i + B}{\sum\limits_i d_i}\right] - \sum_{i=0}^{t-1}\Delta_{Q_i}\end{aligned}$$

where b_i is the coefficient of the multiplicative terms, and B is constant.

Remark VI.1. *Considering our community detection algorithm, if both nodes from an incoming edge exist in different communities then modularity for that component will be 0 as $\delta(c_i, c_j)$ is 0. If both nodes do not exist in any communities or exist in same community then modularity of the graph will be calculated based on the change derived in Lemma VI.1.*

Remark VI.2. *In the graph, if isolated node communities are increased then $\delta(c_i, c_j)$ will be 0 and change in modularity* *will move in negative direction. In other words, the modularity of the whole graph will decrease from the previous point (Figure 7(a)). Here, it can be seen that modularity is decreasing when the number of communities are increasing from 0 to 387 communities for 12500 edges.*

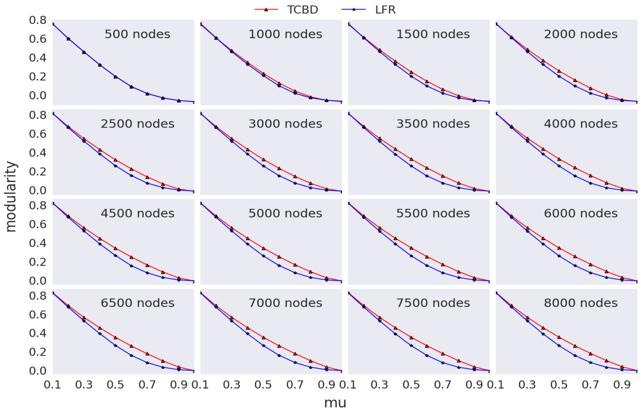

Fig. 5: Modularity analysis with TCBD and Traditional LFR Benchmark algorithm.

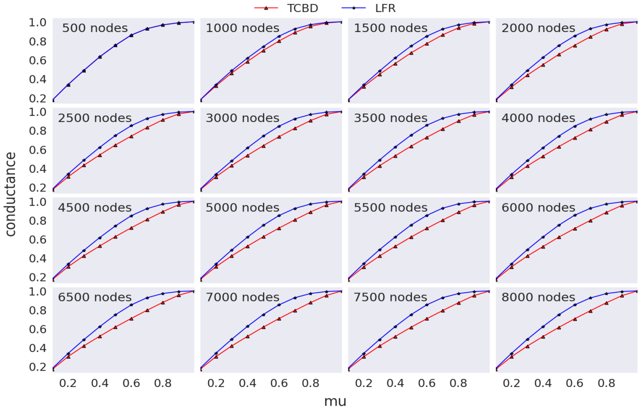

Fig. 6: Conductance analysis with TCBD and Traditional LFR Benchmark algorithm.

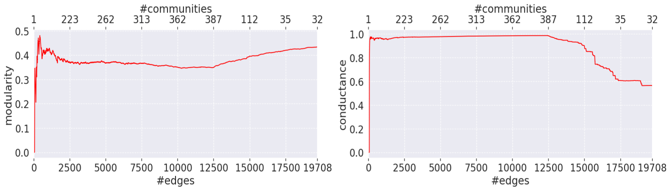

Fig. 7: (a) Modularity (b) Conductance based analysis of proposed algorithm on proposed TCBD.

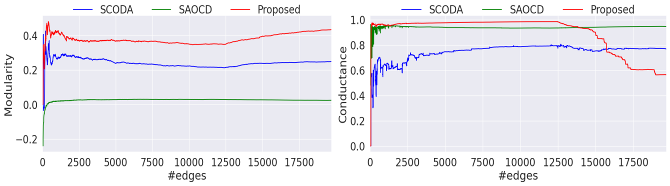

Fig. 8: (a) Modularity (b) Conductance based analysis of proposed algorithm on TCBD.

Remark VI.3. *If ρ_{out} of two communities is greater than the average of ρ_{in} of two communities, a merge will occur, and according to Lemma VI.1, the change in $\delta(c_i, c_j)$ will move in a positive direction. This means that the modularity of the whole graph will increase with every merge of communities. The same is evident in the experiment shown in Figure 7(a), where the modularity increased as communities were merged until the number of communities reached 32 from 387.*

Lemma VI.2. *Given a graph stream S, let $\varphi(c_t)$ represents the conductance of a community c in the graph after time t. For the proposed community detection algorithm, if incoming edge from the stream S is connecting the community c with another community then change in conductance Δ_{φ_t} is $\frac{2|e(c,c)|}{(2|e(c,c)|+|e(c,\bar{c})|)(2|e(c,c)|+|e(c,\bar{c})|+1)}$.*

Proof: *Let an edge e from the stream S is creating outside link of community c to another community. Let $e(c,\bar{c})$ represents the edges going outside the community c, and $e(c,c)$ represents the edges present in the community c. Then, conductance for community c will be:*

$$\begin{aligned}
\varphi(c_t) &= \frac{|e(c,\bar{c})|}{2|e(c,c)|+|e(c,\bar{c})|} \\
\varphi(c_{t+1}) &= \frac{|e(c,\bar{c})|+1}{2|e(c,c)|+|e(c,\bar{c})|+1} \\
\therefore \Delta\varphi_t &= \varphi(c_{t+1}) - \varphi(c_t) \\
&= \frac{|e(c,\bar{c})|+1}{2|e(c,c)|+|e(c,\bar{c})|+1} - \frac{|e(c,\bar{c})|}{2|e(c,c)|+|e(c,\bar{c})|} \\
\Delta\varphi_t &= \frac{2|e(c,c)|}{(2|e(c,c)|+|e(c,\bar{c})|)(2|e(c,c)|+|e(c,\bar{c})|+1)}
\end{aligned}$$

Remark VI.4. *When isolated node communities are increased in the graph then according to Lemma VI.2, overall conductance will increase in positive direction (Figure 7(b)). The same can be validated from Figure 7(b) where conductance is increasing from 0 to 387 communities.*

Lemma VI.3. *Given a graph stream S, let $\varphi(c_t)$ represents the conductance of a community c in the graph after time t. For the proposed community detection algorithm, if incoming edge from the stream S is connecting nodes inside the community c then change in conductance Δ_{φ_t} is $\frac{-2|e(c,\bar{c})|}{(2|e(c,c)|+|e(c,\bar{c})|+2)(2|e(c,c)|+|e(c,\bar{c})|)}$.*

Proof: *Let an edge e from the stream S is connecting two nodes within community c. Let $e(c,\bar{c})$ represents the edges going outside the community c, and $e(c,c)$ represents the edges present in the community c. Then, conductance for community c will be:*

$$\varphi(c_{t-1}) = \frac{|e(c,\bar{c})|}{2|e(c,c)|+|e(c,\bar{c})|}$$

An edge is added inside the community c. Then conductance is,

$$\begin{aligned}
\varphi(c_t) &= \frac{|e(c,\bar{c})|}{2(|e(c,c)|+1)+|e(c,\bar{c})|} \\
\therefore \Delta_{\varphi_t} &= \varphi(c_t) - \varphi(c_{t-1}) \\
&= \frac{|e(c,\bar{c})|}{2(|e(c,c)|+1)+|e(c,\bar{c})|} - \frac{|e(c,\bar{c})|}{2|e(c,c)|+|e(c,\bar{c})|} \\
&= \frac{|e(c,\bar{c})|}{2|e(c,c)|+|e(c,\bar{c})|+2} - \frac{|e(c,\bar{c})|}{2|e(c,c)|+|e(c,\bar{c})|} \\
\Delta\varphi_t &= -\frac{2|e(c,\bar{c})|}{(2|e(c,c)|+|e(c,\bar{c})|+2)(2|e(c,c)|+|e(c,\bar{c})|)}
\end{aligned}$$

Remark VI.5. *If two communities are merged into one community then merge community will have denser links. As a* *result, change in conductance will move in negative direction (Lemma VI.3). In other words, conductance of the whole graph will decrease. From the experimental analysis, it is clear that when communities are getting merged then conductance is moving in negative direction from 387 to 32 communities as shown in Figure 7(b)*

A. Comparative Analysis

We compared our proposed TCBD algorithm with state-of-the-art algorithms SCODA and SAOCD in terms of modularity and conductance. A temporal graph with 1000 nodes and 21000 edges was constructed for the analysis. Our algorithm was evaluated in the online setting, and modularity and conductance were calculated after each edge was received in the stream. Figure 8(a) shows that our algorithm achieved higher modularity compared to other algorithms, except when a large number of single-node communities were established. Similarly, Figure 8(b) shows that our algorithm performed better in terms of conductance than other algorithms.

VII. Experiments and Results

Experiments were conducted on publicly available networks from Stanford Social Network Analysis Project (SNAP [16]) to compare the performance of the proposed algorithm with other state-of-the-art algorithms. The experiments were run on a DGX server with 128 GB system memory. This section presents detailed information about the dataset, results, and corresponding analysis obtained from the experiments.

A. Comparative Algorithms

We checked the performance of our proposed algorithm with following state-of-the-art algorithms:

- **SCODA** [13] is a linear streaming community detection algorithm based on the principle that edges are more likely to connect nodes within the same community than nodes in different communities
- **SAOCD** [32] is an improvement over SCODA that incorporates node contribution and considers how a node's edges change when it moves from one community to another, leading to more accurate network partitioning.

We also compared with the following well known community detection algorithms for static graphs.

- **Louvain** [38] uses hierarchical clustering and combines communities in a recursive manner by executing modularity clustering on condensed networks.
- **Infomap** [6] uses random walk and compresses information using community partition as Huffman code.
- **Walktrap** [5] is also a random walk based community detection algorithm that estimates node similarity using random walks and then clusters the network.

B. Dataset Description

There are different categories of the dataset available in SNAP dataset [16] such as Co-citation Network (DBLP [39]), Co-purchasing Network (Amazon [40]), and Social Network (Orkut, LiveJournal, and Youtube [22]). Dataset with ground truth communities is used in our experiments. Table II lists the properties of these datasets. Note that these datasets do not have any temporal information. In order to feed these data to the proposed and comparing streaming algorithms we assign uniform random timestamps to each edges. However, as the ground truth communities are for the whole graph. We compared the results once all the edges are processed.

While evaluation of the community structure at the end is important, algorithms for streaming community needs to be validated for intermediate results. In other words, any streaming community detection algorithm should provide valid community details anytime an user queries. Static dataset discussed above are not suitable for this. Hence, experiments also performed with data generated with proposed TCBD.

TABLE II: Statistics of real networks from SNAP datasets.

Graph	**Type**	**Nodes**	**Edges**	**Average Degree**	**Ground Truth Communities**
Amazon	Co-purchasing	334,863	925,872	2.76	311,782
DBLP	Co-citation	317,080	1,049,866	3.31	1,449,666
Youtube	Social	1,134,890	2,987,624	2.63	8,455,253
LiveJournal	Social	3,997,962	34,681,189	8.67	137,177
Orkut	Social	3,072,441	117,185,083	38.14	49,732

C. Evaluation Metrics

Effectiveness of our proposed algorithm is checked by three metrics viz 1) Average F1 score [41] 2) Normalized Mutual Information (NMI) [10] 3) Weighted Community Clustering (WCC) [21]. Further execution time and space are used for comparing algorithms.

Consider dividing the graph into N communities , $C = \{C_1, \ldots, C_N\}$. The $F1$-Score and average F1-Score of partition $\widetilde{C} = \left\{\widetilde{C}_1, \ldots, \widetilde{C}_M\right\}$ with respect to C is defined as:

$$F1(\widetilde{C}, C) = \frac{1}{N} \sum_{n=1}^{N} \max_{1 \leq m \leq M} F1\left(\widetilde{C}_m, C_n\right)$$

$$\overline{F1}(\widetilde{C}, C) = (F1(\widetilde{C}, C) + F1(C, \widetilde{C}))/2$$

NMI helps to identify that how much two partitions are similar to each other whereas WCC ensures that communities are cohesive, structured, and well defined. Mathematically both are defined as follows:

$$I_{\text{norm}}(X : Y)(NMI) = \frac{H(X) + H(Y) - H(X, Y)}{(H(X) + H(Y))/2}$$

Where $H(X), H(Y)$ is the entropy of the random variable X, Y associated to the partition $C, \widetilde{C}$, whereas $H(X, Y)$ is the joint entropy.

$$WCC(x, C) = \begin{cases} \frac{t(x,C)}{t(x,V)} \cdot \frac{vt(x,V)}{|C \backslash \{x\}| + vt(x, V \backslash C)} & \text{if } t(x, V) \neq 0 \\ 0 & \text{if } t(x, V) = 0 \end{cases}$$

Where $vt(x, C)$ is the number of vertices in C that form at least one triangle with x and $t(x, C)$ is the number of triangles that vertex x surrounded with vertices in C. Finally, the WCC of graph $G = \{C_1, \ldots, C_n\}$ such that $(C_1 \cap \ldots \cap C_n) = \phi$ is:

$$WCC(G) = \frac{1}{|V|} \sum_{i=1}^{n} (|C_i| . WCC(C_i))$$

Where $WCC(C)$ is the average of $WCC(x, C)$.

TABLE III: Comparison with state-of-the-art algorithms in terms of 1) Nomalized Mutual Information (NMI) 2) F1-Score 3) Weighted Community Clustering (WCC) 4) Execution Time (ET) (in seconds) and 5) Execution Memory (EM) (in GBs).

Graph	**Amazon**					**DBLP**					**Youtube**					**LiveJournal**					**Orkut**				
/Algorithm	NMI	F1	WCC	ET	EM	NMI	F1	WCC	ET	EM	NMI	F1	WCC	ET	EM	NMI	F1	WCC	ET	EM	NMI	F1	WCC	ET	EM
Infomap	.16	.31	.00	47.6	.56	.00	.09	.01	45.5	.60	.00	.01	.00	191.4	2.02	.01	.04	.00	2908.3	14.51	.00	.04	.00	4165.2	101.59
Louvian	.14	.28	.01	93.5	.61	.06	.13	.01	202.9	.67	.00	.00	.00	436.4	2.19	.02	.08	.01	12111.4	15.02	-	-	-	-	-
Walktrap	**.27**	**.44**	.13	1291.5	.42	**.10**	**.29**	**.16**	2747.6	.47	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
SCODA	.11	.37	.09	3.4	.19	.05	.23	.10	3.8	.21	**.07**	**.22**	.01	16.2	**.67**	**.06**	**.23**	.01	190.7	**4.96**	.17	.37	.00	**695.9**	**14.39**
SAOCD	.16	.40	.11	8.2	.23	.07	.25	.12	9.3	.27	.04	.15	.00	31.3	.80	.03	.14	.01	350.4	8.23	.10	.30	.00	1677.3	27.19
Proposed	.17	.41	**.20**	**3.2**	**.18**	**.10**	**.29**	**.16**	**3.67**	**.20**	.05	.20	**.02**	**15.4**	.68	.03	.16	**.03**	**183.2**	6.34	**.30**	**.45**	**.03**	711.9	17.83

D. Results

We have analyzed the results of proposed algorithm in terms of evaluation metrics.

1) Detection Score: Table III shows the NMI and F1-score results of the proposed algorithm and other community detection algorithms on SNAP datasets. Each algorithm was run at least 15 times by randomly shuffling the edge list before each run. The best result from each trial was then reported in the table. Algorithms took more than 5 hours and produced no results, indicated by a dash in the table. The proposed algorithm outperformed all other algorithms in Orkut networks and except Walktrap in Amazon, produced comparable results with SCODA in Youtube and LiveJournal, and matched the result of Walktrap in DBLP.

Table III also shows the WCC results, indicating that the proposed algorithm outperforms other sota algorithms in terms of making communities cohesive and well structured.

2) Execution Time and Memory: Table III compares the execution time and memory used by our proposed algorithm with other methods. Our algorithm outperforms other methods in all the graphs except Orkut, taking only 3 seconds for small graphs and 711 seconds for the largest graph (Orkut) with billions of edges. In terms of memory, the proposed algorithm takes lowest peak memory in two datasets while for others it takes second lowest execution space.

E. Effect of Different Condition of Merge

We varied the fraction threshold α (Algorithm 1: Line 18) from 0.1 to 1.0 and evaluated it on the SNAP datasets. The results are shown in Figure 9(a) and 9(b), where a 0.6 threshold value gave the highest NMI and F1-score for all networks except Amazon. In Amazon, the highest NMI was achieved at a threshold value of 0.1, indicating the possibility of smaller-sized communities in the Amazon network. These results suggest that varying the α can improve the NMI and F1-score of the network when ground truth is available.

F. Correctness for Streaming Queries

We tested our proposed algorithm on a streaming environment using a TCBD graph with mixing parameter μ values of 0.2 and 0.4. To compare the performance of streaming-based community detection algorithms, we generated a graph and added 500 nodes per iteration until it reached 8000 nodes. Figure 10 illustrates the results, and our proposed algorithm performed significantly better than the others.

VIII. Discussions and Conclusion

In the paper, we developed a small space Community Sketch to preserve the communities structure in the graph with $O(|V| + |C|^2)$ space requirement, where $|C|$ is the number of communities in the graph. The community sketch stores distinct communities with a forest and uses a triangular matrix of counters to store the number of edges between communities. The data structure can accommodate any distinct streaming community detection algorithms by adjusting the counters. The query time to get the community structure is constant.

We proposed a streaming community detection algorithm using the community sketch data structure. The running time of the algorithm is linear to the number of edges, i.e., $O(|E|)$. It processes each edge only once. We assume that edges in the stream are distinct. However, in real graph streams, edges can be repeated; this can be thought of as a future research problem. We have also analyzed the change in modularity and conductance during the execution of the algorithm.

The algorithm is shown to work with different social networks, including large-scale networks with billions of edges. A comparative analysis with other state-of-the-art streaming community detection algorithms and well-known static community detection algorithms is presented. In terms of evaluation metrics, the proposed algorithm is seen to produce comparable or higher results. Our algorithm exhibits a significant improvement in Weighted Community Clustering (WCC) compared to other methods, indicating its ability to produce highly cohesive and well structured communities.

Further, we proposed Temporal Community Benchmark Dataset (TCBD) algorithm that generates graph with temporal information and ground truth Communities. The graph generated using TCBD contains several snapshots in different times contains incremental community labels. Although, these snapshots do not truly provide a streaming graph, it provides dynamic community labels. With modifying certain parameters, streaming graph can be generated, however in that case number of communities will be static. Dealing with these limitations are kept as a future work. Inspite the limitations, this benchmark algorithm fills the gap in the research on Community Detection in graph streams and may provide a data set to verify any dynamic or streaming community detection algorithms.

References

[1] A.-L. Barabási, "Network science," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 371, no. 1987, p. 20120375, 2013.

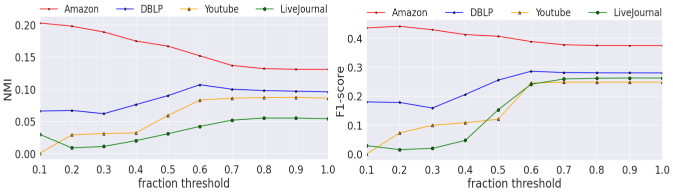

Fig. 9: Effect of different α value on (a) NMI (b) F1-score.

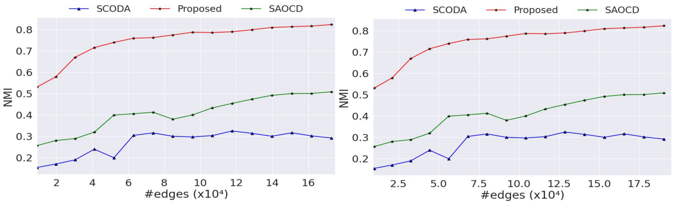

Fig. 10: NMI analysis on stream coming in different timestamps on TCBD Network (a) $\mu = 0.2$ (b) $\mu = 0.4$.

[2] M. Girvan and M. E. J. Newman, "Community structure in social and biological networks," *Proc. of the National Academy of Sciences*, vol. 99, no. 12, pp. 7821–7826, 2002.

[3] M. E. J. Newman and M. Girvan, "Finding and evaluating community structure in networks," *Phys. Rev. E*, vol. 69, p. 026113, Feb 2004.

[4] M. E. J. Newman, "Modularity and community structure in networks," *Proc. of the National Academy of Sciences*, vol. 103, no. 23, pp. 8577–8582, 2006.

[5] P. Pons and M. Latapy, "Computing communities in large networks using random walks," in *Computer and Information Sciences - ISCIS 2005*, p. Yolum, T. Güngör, F. Gürgen, and C. Özturan, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 284–293.

[6] M. Rosvall and C. T. Bergstrom, "Maps of random walks on complex networks reveal community structure," *Proc. of the National Academy of Sciences*, vol. 105, no. 4, pp. 1118–1123, 2008.

[7] D. A. Spielman and S.-H. Teng, "Spectral partitioning works: Planar graphs and finite element meshes," *Linear Algebra and its Applications*, vol. 421, no. 2, pp. 284–305, 2007, special Issue in honor of Miroslav Fiedler.

[8] U. von Luxburg, "A tutorial on spectral clustering," *Statistics and Computing*, vol. 17, pp. 395–416, 2007.

[9] X. Xu, N. Yuruk, Z. Feng, and T. A. J. Schweiger, "Scan: A structural clustering algorithm for networks," in *Proc. of the 13th ACM SIGKDD International Conf. on Knowledge Discovery and Data Mining*, ser. KDD '07. New York, NY, USA: ACM, 2007, p. 824–833.

[10] A. Lancichinetti, F. Radicchi, and J. J. Ramasco, "Statistical significance of communities in networks." *Physical review. E, Statistical, nonlinear, and soft matter physics*, vol. 81 4 Pt 2, p. 046110, 2010.

[11] S. Yazdanparast, T. C. Havens, and M. Jamalabdollahi, "Soft overlapping community detection in large-scale networks via fast fuzzy modularity maximization," *IEEE Transactions on Fuzzy Systems*, vol. 29, no. 6, pp. 1533–1543, 2021.

[12] J. Shi, L. Dhulipala, D. Eisenstat, J. Łącki, and V. Mirrokni, "Scalable community detection via parallel correlation clustering," *Proc. VLDB Endow.*, vol. 14, no. 11, p. 2305–2313, oct 2021.

[13] A. Hollocou, J. Maudet, T. Bonald, and M. Lelarge, "A linear streaming algorithm for community detection in very large networks," 2017.

[14] P. Liakos, A. Ntoulas, and A. Delis, "Coeus: Community detection via seed-set expansion on graph streams," in *2017 IEEE International Conf. on Big Data (Big Data)*. Boston: IEEE, 2017, pp. 676–685.

[15] P. Liakos, K. Papakonstantinopoulou, A. Ntoulas, and A. Delis, "Dices: Detecting communities in network streams over the cloud," in *2019 IEEE 12th International Conf. on Cloud Computing (CLOUD)*. Italy: IEEE, 2019, pp. 301–310.

[16] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford large network dataset collection," Jun. 2014.

[17] A. Lancichinetti, S. Fortunato, and F. Radicchi, "Benchmark graphs for testing community detection algorithms," *Phys. Rev. E*, vol. 78, p. 046110, Oct 2008.

[18] S. Fortunato, "Community detection in graphs," *Physics Reports*, vol. 486, no. 3, pp. 75–174, 2010.

[19] J. Shi and J. Malik, "Normalized cuts and image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 888–905, 2000.

[20] Y. Yang, M. Wang, D. Bindel, and K. He, "Streaming local community detection through approximate conductance," 2021.

[21] A. Prat-Pérez, D. Dominguez-Sal, J. M. Brunat, and J.-L. Larriba-Pey, "Shaping communities out of triangles," in *Proc. of the 21st ACM International Conf. on Information and Knowledge Management*, ser. CIKM '12. New York, NY, USA: ACM, 2012, p. 1677–1681.

[22] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," *Knowledge and Information Systems*, vol. 42, pp. 181–213, 2012.

[23] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, p. P10008, oct 2008.

[24] J. J. Whang, D. F. Gleich, and I. S. Dhillon, "Overlapping community detection using seed set expansion," in *Proc. of the 22nd ACM International Conf. on Information &; Knowledge Management*, ser. CIKM '13. New York, NY, USA: ACM, 2013, p. 2099–2108.

[25] G. Palla, I. Derényi, I. J. Farkas, and T. Vicsek, "Uncovering the overlapping community structure of complex networks in nature and society," *Nature*, vol. 435, pp. 814–818, 2005.

[26] S. Moon, J.-G. Lee, M. Kang, M. Choy, and J. woo Lee, "Parallel community detection on large graphs with mapreduce and graphchi," *Data & Knowledge Engineering*, vol. 104, pp. 17–31, 2016.

[27] S.-H. Bae, D. Halperin, J. D. West, M. Rosvall, and B. Howe, "Scalable and efficient flow-based community detection for large-scale graph analysis," *ACM Trans. Knowl. Discov. Data*, vol. 11, no. 3, mar 2017.

[28] U. N. Raghavan, R. Albert, and S. Kumara, "Near linear time algorithm to detect community structures in large-scale networks," *Physical review E*, vol. 76, no. 3, p. 036106, 2007.

[29] H. Li, R. Zhang, Z. Zhao, and X. Liu, "Lpa-mni: An improved label propagation algorithm based on modularity and node importance for community detection," *Entropy*, vol. 23, no. 5, 2021.

[30] R. Sun, C. Chen, X. Wang, Y. Zhang, and X. Wang, "Stable community detection in signed social networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 10, pp. 5051–5055, 2022.

[31] S. Sabour and A. a. Moeini, "Creating a maximal clique graph to improve community detection in scoda and oslom algorithms," *International Journal of Information and Communication Technology Research*, vol. 11, no. 4, 2019.

[32] H. Li, F. Chen, and J. Zhang, "A streaming-based algorithm for overlapping community detection," in *2020 IEEE 6th International Conf. on Computer and Communications (ICCC)*. China: IEEE, 2020, pp. 1925–1930.

[33] P. Liakos, K. Papakonstantinopoulou, A. Ntoulas, and A. Delis, "Rapid detection of local communities in graph streams," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 5, pp. 2375–2386, 2022.

[34] A. E. Saríyüce, B. Gedik, G. Jacques-Silva, K.-L. Wu, and U. V. Çatalyürek, "Streaming algorithms for k-core decomposition," *Proc. VLDB Endow.*, vol. 6, no. 6, p. 433–444, apr 2013.

[35] A. E. Sarıyüce, B. Gedik, G. Jacques-Silva, K.-L. Wu, and U. V. Çatalyürek, "Incremental k-core decomposition: Algorithms and evaluation," *The VLDB Journal*, vol. 25, no. 3, p. 425–447, jun 2016.

[36] Y. Wu, J. Tardos, M. Bateni, A. Linhares, F. M. Goncalves de Almeida, A. Montanari, and A. Norouzi-Fard, "Streaming belief propagation for community detection," *Advances in Neural Information Processing Systems*, vol. 34, pp. 26 976–26 988, 2021.

[37] J. E. Hopcroft and J. D. Ullman, "Set merging algorithms," *SIAM Journal on Computing*, vol. 2, no. 4, pp. 294–303, 1973.

[38] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, p. P10008, 2008.

[39] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan, "Group formation in large social networks: Membership, growth, and evolution," in *Proc. of the 12th ACM SIGKDD International Conf. on Knowledge Discovery and Data Mining*, ser. KDD '06. New York, NY, USA: ACM, 2006, p. 44–54.

[40] J. Leskovec, L. A. Adamic, and B. A. Huberman, "The dynamics of viral marketing," *ACM Trans. Web*, vol. 1, no. 1, p. 5–es, may 2007.

[41] J. Yang and J. Leskovec, "Overlapping community detection at scale: A nonnegative matrix factorization approach," in *Proc. of the Sixth ACM International Conf. on Web Search and Data Mining*, ser. WSDM '13. New York, NY, USA: ACM, 2013, p. 587–596.