Rapid Detection of Local Communities in Graph Streams

Panagiotis Liakos, Katia Papakonstantinopoulou, Alexandros Ntoulas, and Alex Delis

Abstract—We examine the problem of uncovering communities in complex real-world networks whose elements and their respective associations manifest as streams of data. Community detection is applied in emerging computational environments and concerns critical applications in diverse areas including social computing, web analysis, IoT and biology. Despite the already expended related research efforts, the task of revealing the community structure of massive and rapidly-evolving networks remains very challenging. More specifically, there is an emerging need for online approaches that ingest graph data as a stream. In this paper, we propose a streaming-graph community-detection algorithm that expands seed-sets of nodes to communities. We consider an online setting and process a stream of edges while aiming to form communities on-the-fly using partial knowledge of the graph structure. We use space-efficient structures to maintain very limited information regarding the nodes of the graph and the sought communities, so as to effectively process large scale networks. In addition to our novel streaming approach, we develop a technique that increases the accuracy of our algorithm considerably and additionally propose a new clustering algorithm that allows for automatically deriving the size of the communities we seek to detect. Using ground-truth communities for a wide range of large real-word and synthetic networks, our experimental evaluation shows that our approach does achieve accuracy comparable, and oftentimes better, to the state-of-the-art non-streaming community detection algorithms. More importantly, we attain significant improvements in both execution time and memory requirements.

Index Terms—Local Community Detection, Graph Streams, Seed-set Expansion, Social Networks.

1 Introduction

Revealing the community structure of networks representing entities in various domains is a challenge that garners the interest of both industry and academia. Despite their size, networks exhibit a high level of order and organization, a property frequently referred to as community structure [1]. Nodes tend to organize into densely connected groups that exhibit weak ties with the rest of the graph. We refer to such groups as communities, whereas the task of identifying them is termed *community detection*.

Community detection is a fundamental problem in the study of networks and becomes more relevant with the prevalence of online social networking services such as LinkedIn and Facebook. Understanding network communities leads to invaluable insights around the functioning of many integral systems, in areas such as social computing, web analysis, IoT and biology. In a social context, identifying the communities of individuals enables us to perform recommendations for new connections. Moreover, by uncovering the membership of an individual to various organizational groups, we can provide more informative and engaging social network feeds. Community detection can be also successfully applied to numerous other types of networks, such as the World Wide Web or biological networks. In the context of the World Wide Web, we are often interested in identifying topic-focused communities and tracking their evolution in time [2]. Biological networks comprise among others neural networks, food webs, and metabolic networks [1], on which we are particularly interested in inferring their functional modules such as cycles and pathways [2].

In the last two decades, a plethora of community detection methods has been proposed. Initially, the focus has been on non-overlapping communities [3], [4], [5], [6]. More recent approaches, however, allow for nodes to belong to more than one community [7], [8], [9], [10], [11], [12]. Still, these approaches focus on the *entire* graph structure and do not scale with regards to both execution time and memory consumption; hence, they are also not applicable to the massive and dynamically formed graphs of the Big Data era. Recent efforts manage to scale as far as execution time is concerned by focusing on the *local* structure and expanding exemplary seed-sets into communities [13], [14], [15], [16]. Such a semi-supervised learning task can be applied to numerous real world applications, e.g., given a few researchers focusing on "Data Engineering" we can use a citation network to detect their colleagues in the same field. However, the space requirements of such algorithms rapidly become a concern due to the unprecedented size now reached by real-world graphs. The latter have become difficult to represent in-memory even in a distributed setting [17].

An increasingly popular approach for massive graph processing is to consider a *data stream model*, in which the stream comprises the edges of a graph [18], [19], [20]. The challenge here is to process this graph stream based exclusively on single-pass access to the stream and limited working memory [19]. This is a new direction in the field of community detection and to the best of our knowledge

This work has been partially supported by the EU H2020 "GALENA" grant with agreement n. 641515.

- *P. Liakos, A. Ntoulas, and A. Delis are with the Department of Informatics and Telecommunications, University of Athens, Athens 15703, Greece. E-mail: {p.liakos, antoulas, ad}@di.uoa.gr.*
- *K. Papakonstantinopoulou is with the Department of Informatics, Athens University of Economics and Business, Athens GR 10434, Greece. Email: katia@aueb.gr*

Manuscript received January 7, 2020; revised 9/6/2020

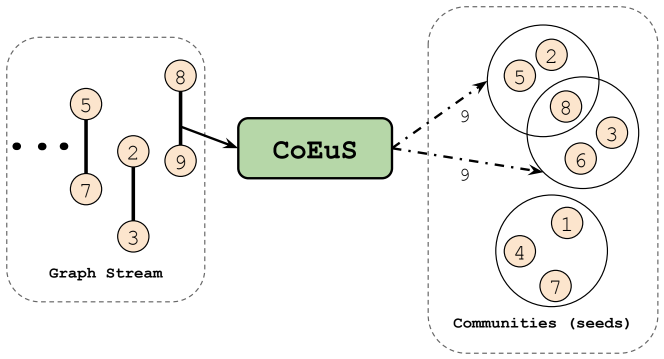

Fig. 1: A stream comprising the edges of an undirected graph and a set of communities initialized with a few seed nodes. For every edge of the stream we wish to evaluate whether the adjacent nodes belong to the communities we examine.

no prior approach has considered such a setting without imposing restrictions on the order in which edges are made available [21], [22]. In this paper, we propose COEUS,[1] a novel *online* local community detection algorithm that is fully applicable on graph streams.

Figure 1 depicts such a graph stream whose edges arrive at no particular order. COEUS is initialized with seed-sets of nodes that define different communities, such as the three sets depicted with the circles of Figure 1. As edges arrive, we can process them but we cannot afford to keep them all in-memory. Therefore, COEUS maintains rather limited information about the adjacent nodes of each edge and their participation in the communities in question. This information is kept using probabilistic data structures to further reduce the memory requirements of our algorithm. In addition to our original approach for community detection in graph streams, we propose two algorithms to enhance the effectiveness of COEUS. The first focuses on better quantifying the quality of each edge w.r.t. a community. The second is a novel clustering algorithm that allows for automatically determining the size of the resulting communities, in spite of the absence of the graph structure.

Our experimental results derived using various large-scale real-world and synthetic graphs show that COEUS is extremely competitive with regard to *accuracy* compared to approaches that employ the *entire graph* structure and thus cannot operate on graph streams. More specifically, COEUS can process with just a few MBs, graphs that prior approaches fail to handle on a machine with 16GB of RAM. In addition, COEUS derives the communities in question inordinately faster. For instance, we demonstrate that COEUS is almost 30 times faster for the largest graph we could process with previously suggested approaches. More importantly, COEUS is able to return its resulting communities *on demand* at any time while we *ingest the graph as a stream*. This is particularly important, as no other approach is able to update communities while new edges arrive without additional *significant* computational cost, regardless of the memory space we can afford to expend.

In summary, we make the following contributions:

- We propose COEUS, a novel online community detection algorithm that operates on a graph stream. To the best of our knowledge, this is the first community detection algorithm that uses space sublinear to the number of edges and does not impose any restrictions on the order in which edges arrive in the stream.
- We develop a variation of our algorithm to better quantify the quality of each edge w.r.t. a community and verify that it improves the accuracy of COEUS impressively.
- We suggest a novel clustering algorithm that allows for automatically determining the size of the resulting communities of COEUS.
- We experimentally evaluate the accuracy of our algorithm and show that COEUS is extremely competitive with prior approaches that cannot operate on graph streams and instead need to load the complete graph in memory. In addition, we show that both the execution time and space requirements of COEUS are astonishingly low.

2 Community Detection via Seed-Set Expansion on Graph Streams

In this section, we first formulate the challenge we address in this work. Then, we discuss the space requirements of our algorithm, and present our novel approach for streaming community detection. Lastly, we propose two enhancements to our algorithm, that greatly improve its effectiveness and efficiency.

2.1 Problem formulation

Consider a streaming sequence of unique unordered pairs of nodes $e = \{u, v\}$. Over time such a stream $S = \langle e_1, e_2, \ldots, e_m \rangle$ naturally defines an undirected unweighted graph $G = \{V, E\}$, where V is a set of vertices $\{v_1, v_2, \ldots, v_n\}$ and E is a set of undirected edges $\{e_1, e_2, \ldots, e_m\}$. Given a few exemplary members of some community, i.e., a community seed-set $K = \{k_1, k_2, \ldots, k_l\}$ where each $k_i \in V$, our goal is to extend K to the target community C. Figure 1 shows such a graph stream with three visible arriving edges, and three seed-sets that are to be extended to communities.

A community is generally thought of as a set of graph nodes that are tightly connected to each other and exhibit limited cohesion with the rest of the graph's nodes [5]. However, there is no universal definition of what communities are, and thus, there exists a plethora of different approaches in detecting them. A widely used quality function in the field of community detection is the *conductance* of a community [9], [10], [15], [23]. More specifically, the conductance $\phi(C)$ of a community C is formally defined as:

$$\phi(C) = \frac{\texttt{adj}(C, V \setminus C)}{\min\{\texttt{adj}(C, V), \texttt{adj}(V \setminus C, V)\}}, \qquad (1)$$

where $\texttt{adj}(A, B) = |\{(u, v) \in E : u \in A, v \in B\}|$.

[1] COEUS stands for Community detection via seed-set Expansion on graph Streams. In Greek mythology Coeus was the Titan of intellect, the axis of heaven around which the constellations revolved and probably of heavenly oracles.

Several methods attempt to detect communities exhibiting low conductance, in an effort to come up with a set of nodes with a limited number of ties to nodes outside the community. Moreover, there do exist approaches that offer in-memory approximations of evolving subgraphs that can be used in computing and tracking the conductance of these subgraphs as edge-activations arrive [24]. However, local community detection approaches based on conductance [15] need to calculate the corresponding conductance for each of the possible subsets of an expanded community. Such an approach is impractical in a streaming setting.

We introduce here the *community participation* $\mathtt{cp}(u,t)$ of a node u in a community at time-step t, as a means to pragmatically measure node's u participation level in this community. We assume that the stream elements arrive in successive and distinct time-steps.

Definition 1. The community participation of node u in community C at time-step t is defined as:

$$\mathtt{cp}(u,t) = \begin{cases} 1, & \text{if } u \in K \\ \dfrac{|\{(u,v) \in S_t : v \in C_{t-1}\}|}{|S_t|}, & \text{otherwise} \end{cases} \tag{2}$$

where K is the seed-set for target community C, S_t is the set of elements of the stream S that have arrived at time-step t and C_{t-1} comprises the set of nodes in community C at time-step $t-1$.

We employ $\mathtt{cp}(u,t)$ to approximate, for each node, the fraction of its neighbors in the graph that are part of a community we seek to expand. Our intuition is that including nodes exhibiting high values of $\mathtt{cp}$ to a community C will result to a low value of *conductance* for the community. To this end, we employ Eq. (2) to detect communities. We note, however, that the use of a particular quality function, such as conductance or community participation, does not hinder in any way the evaluation of our approach against community detection methods using different quality functions. Such an evaluation is possible, as there exist publicly available networks with ground-truth communities. Our experimental setting features numerous such networks that allow us to verify the efficiency of different approaches.

2.2 Space complexity

Real-world networks may reach voluminous sizes that cause serious challenges to classical analysis algorithms [18]. One approach to process such networks is using graph stream algorithms. The latter handle a stream comprising the edges of the graph –in the order in which these edges arrive over time– using *limited memory*.

In the context of community detection, prior streaming approaches are shown to successfully reveal the community structure of graph streams with limited memory requirements [21], [22]. However, all previous approaches impose additional constraints on the order in which the edges of the stream arrive. In particular, Yun et al. [22] consider a data stream model in which rows of the adjacency matrix of the graph are *revealed sequentially*. In such a setting we must be aware at any moment of all neighbors of certain nodes. Thus, [22] applies community detection with partial information as the subgraphs are revealed. Memory requirements are kept low as at each step all information that was made available in earlier steps can be discarded. `SCoDa` [21] considers a setting in which the edges of the graph stream arrive as if we picked them uniformly at random. This allows for estimating whether a newly arriving edge is an intra-community or an inter-community edge and enables `SCoDa` to achieve space complexity that is linear to the number of the graph's nodes. However, picking an edge of the graph uniformly at random requires that we already possess the graph in its entirety; this assumption is not true for graph streams.

In this paper, we consider a more practical scenario of a streaming setting in which the edges of a graph arrive at no particular order. Thus, we cannot discard information in ways similar to the techniques in [21], [22]. Our approach aims at estimating the participation level of each node of the graph in each of the communities we examine, using Eq. (2). In this context, we need to keep track of the following information as we process a graph stream:

1) **communities**: the set of nodes that comprise each community we examine,
2) **degrees**: the degree of each node in the graph, i.e., the total number of nodes each node in the graph is adjacent to, and
3) **community degrees**: the degree of each node in the subgraph induced by each community we examine, considering the communities in the form they have at the time we process the stream.

Essentially, if $|C'|$ is the number of communities we examine, the above information can be kept in-memory using $|C'|$ sets (one set for each community we examine), and $|V|(|C'|+1)$ integers. Specifically, for each node in the graph we need one integer to hold its degree and another to hold its community degree in each community we examine, which gives a total of $|C'|+1$ integers. Given that the number of communities we examine can be very large, we opted for using COUNT-MIN sketches to store the $|V|(|C'|+1)$ integers efficiently in main memory.

The COUNT-MIN sketch [25] is a well-known sublinear space data structure for the representation of high-dimensional vectors. COUNT-MIN sketches allow fundamental queries to be answered efficiently and with strong accuracy guarantees. They are particularly useful for summarizing data streams as they are capable of handling updates at high rates. A COUNT-MIN sketch uses a $d \times w$ matrix where $w = \lceil \frac{e}{\epsilon} \rceil$, $d = \lceil ln \frac{1}{\delta} \rceil$, e is Euler's number (the base of the natural logarithm), and the error in answering a query is within a factor of ϵ with probability δ. A total of d pairwise independent hash functions is also used, each one associated with a row of the array.

Figure 2 illustrates the update process of a COUNT-MIN sketch for our specific problem. Consider that an edge (u,v) arrives at the stream and as $v \in C$ we need to increase the number of adjacent nodes u has in community C. Thus, we form a unique *pair* using the labels of the community (C) and the node (u) and create an update $(C{:}u, 1)$, indicating that the count of $C{:}u$ should be incremented by 1. The array `count` is updated as follows: For each row i of `count` we apply the corresponding hash function to obtain a column index $j = h_i(C{:}u)$ and increment the respective value of the

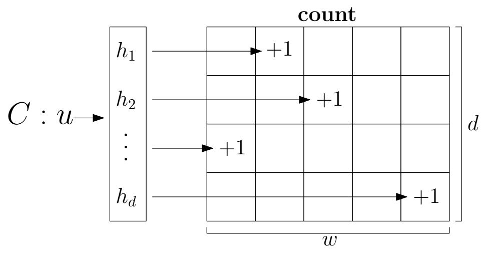

Fig. 2: COUNT-MIN Sketch update process.

array by 1, i.e., $\texttt{count}[i, j]$+=1. This allows us to retrieve at any time for a pair $C{:}u$ an (over)estimation of its count $a_{\hat{C:u}}$, using the least value in the count array for $C{:}u$, i.e., $a_{\hat{C:u}} = \min_i\{\texttt{count}[i, h_i(C{:}u)]\}$.

2.3 Our CoEuS Algorithm for Streaming Community Detection

In this section, we detail our algorithm for streaming community detection, termed COEUS. Algorithm 1 provides the pseudocode of COEUS.

Input/Output: COEUS takes two parameters as input. The first is a family of community seed-sets $K' = \{K_1, K_2, ..., K_s\}$ and each $K_i = \{k_1, k_2, \ldots, k_l\} \subseteq V$. The second is a stream $S = \langle e_1, e_2, \ldots, e_m\rangle$, where each $e_i \in E$, and E is the set of edges of the undirected graph $G = \{V, E\}$ resulting from S. COEUS processes the edges of the graph stream to extend each of the seed-sets in K' to a community. Thus, the output of COEUS is the set of communities $C' = \{C_1, C_2, ..., C_s\}$, with community C_i corresponding to seed-set K_i. The resulting communities are accessible *on-demand* at all times as we process the stream.

Initialization: The first step of COEUS is to initialize the two COUNT-MIN sketches (Lines 2-3) and a set C' that will hold references to the communities and will be the output of the algorithm (Line 4). Next, we initialize the targeted communities using the seed-sets (Lines 5-10). This phase creates an additional set for each of the community seed-sets, to hold the nodes of the respective communities. The seed-sets and the community sets enable us to query efficiently at any time whether a node is a seed or a member of a community. Using Figure 1 as an example, consider that we wish to detect three communities. COEUS is initiated with three seed-sets that designate these communities, namely $\{2, 5, 8\}$, $\{3, 6, 8\}$, and $\{1, 4, 7\}$. In this setting, COEUS creates three community sets that comprise these nodes, as we are certain that the seeds of a community are part of the community (Line 8). Additionally, we maintain an index I that allows for efficient retrieval of the communities a node is part of (Line 9).

Stream processing: After initializing the communities, COEUS is ready to process the elements of the stream (Lines 11-28). Our working assumption is that maintaining the whole graph is prohibitive. Instead, we focus on the degree of each node in the graph, the node's degree in each community, and the nodes that comprise each community. Algorithm 1 additionally maintains the communities each node is part of, which helps effectively eliminate computation overheads; however, this information is not necessary for COEUS.

Algorithm 1: COEUS(K', S)

```
input      : Set of community seed-sets $K'$,
             graph stream $S$.
output     : Set of communities $C'$.
parameters : Window size $W$, pruning size $s$.
1  begin
2      degree$_V$ ← $CMS$;
3      degree$_C$ ← $CMS$;
4      $C'$ ← {};
5      foreach $K \in K'$ do
6          $C$ ← {};
7          foreach $k \in K$ do
8              $C \leftarrow C \cup \{k\}$;
9              $I[k] \leftarrow I[k] \cup \{C\}$;
10         $C' \leftarrow C' \cup \{C\}$;
11     processedElements ← 0;
12     while $\exists(u, v) \in S$ do
13         degree$_V[u]$ += 1;
14         degree$_V[v]$ += 1;
15         foreach $C \in (I[u] \cup I[v])$ do
16             if $u \in C$ then
17                 degree$_C[v]$ += 1;
18             if $v \in C$ then
19                 degree$_C[u]$ += 1;
20             if $u \in C$ then
21                 $C \leftarrow C \cup \{v\}$;
22                 $I[v] \leftarrow I[v] \cup \{C\}$;
23             if $v \in C$ then
24                 $C \leftarrow C \cup \{u\}$;
25                 $I[u] \leftarrow I[u] \cup \{C\}$;
26             processedElements += 1;
27             if processedElements mod $W == 0$ then
28                 $C$ ←pruneComm($C, s$, degree$_V$, degree$_C$);
29     return C'
```

For each incoming edge of the stream, we first increment the degree of each of the adjacent nodes in the graph by 1 (Lines 13-14). Then, for each community that the adjacent nodes are part of (Line 15), we examine whether each of the two involved nodes is a member of the community (Lines 16 & 18). If this is the case, we increment the community degree of the other node (Lines 17 & 19). After we update the community degrees, we can examine whether we should add the node to the community (Lines 21 & 24), and the community to the node's communities (Lines 22 & 25). Note that we cannot merge Line 16 with Line 20 nor Line 18 with Line 23, as this would alter the branch evaluations.

Going back to the example of Figure 1, with the arrival of edge $\{9, 8\}$, COEUS will first increment the degree of both nodes 8 and 9 by 1. Then, COEUS will examine the communities that nodes 8 and 9 belong to. Node 8 is featured in two communities whereas node 9 is not included in any. Therefore, COEUS will increment the community degree of node 9 by 1 for both communities node 8 is part of. In addition, COEUS will add node 9 to both communities that node 8 belongs to, and the two communities will be added to the community index of node 9.

As the diameters that real-world networks exhibit are small and in many cases decrease as the network grows [26], the communities COEUS detects through the above process often grow considerably in size. We wish to focus on nodes that are tightly connected to each other for each community. To this end, we additionally consider a window of size

Algorithm 2: pruneComm

```
Function pruneComm ($C$, $s$, $degree_V$, $degree_C$)
    minheap ← [];
    foreach $v \in C$ do
        cp ← $\frac{\mathtt{degree}_C[v]}{\mathtt{degree}_V[v]}$;
        if size(minheap) < $s$ then
            minheap.push($v$, cp);
        else if cp > minheap[0] then
            minheap.pop();
            minheap.push($v$, cp);
    return set(minheap);
```

W. During a window, the communities may grow freely in size, as new edges arrive. When the window closes, CoEuS prunes all communities and keeps only the s most *highly involved* nodes of each community (Lines 27-28).

The above process is detailed by Algorithm 2. Here, the *pruneComm* function uses Eq. 2 to evaluate each node's participation level to community C. For each node $v \in C$ we calculate cp for the current time-step (Line 4). Then, we use a min-heap to hold the nodes with the highest community participation values. If the size of the min-heap is currently below s, i.e., the size at which we want to prune the community, we push the node and its community participation value to the min-heap (Lines 5-6). Otherwise, we examine whether the community participation value of the current node is higher than that of the minimum value in the min-heap (Line 7). If so, we pop the latter out of the min-heap, and push in the current node (Lines 8-9). The function outputs a set comprising the nodes that remained in the min-heap after examining all the nodes of the community (Line 10). For brevity, we omit the respective updates to index I that allows for efficient retrieval of the communities each node participates in. CoEuS prunes communities to a size $s = 100$, as related studies state that quality communities do not surpass 100 nodes [27]. This value of parameter s can be configured depending on the size of targeted communities. Moreover, CoEuS uses a window of 10,000 edges, a value derived via extensive exploratory testing that consistently works well, as we discuss later on.

Termination: CoEuS can be stopped at any time, as the member nodes of each community are available at any moment. The output of Algorithm 1 is the set C' that holds references to all targeted communities that are updated during the stream processing. All community nodes are associated with a community participation value that CoEuS may include in its output. The higher this value is, the more certain we are that the respective node is part of a community. In the pseudocode of Algorithm 1, we consider a finite stream and so, CoEuS terminates when all elements of the stream have been processed. Evidently, CoEuS can handle infinite streams as well.

2.4 Reckoning in edge quality w.r.t. each community

Algorithm 1 examines for every edge of a graph stream whether its two adjacent nodes belong to a community. If this is the case, CoEuS increments the respective community degree of the other node by 1. This procedure takes into consideration only the number of internal links each node has in a community. All nodes included in a community provide increments of 1 to all of their adjacent nodes, regardless of *how well-established the former are* in the community.

Proposition 1. The sum of increments some node u will contribute to the community degrees of other nodes for some community C, given that window $W \to \infty$, depends solely on the number of edges incident on u that are processed after u has become part of C.

Proof: Assume that there is a set of n edges that are incident on u and are processed after u is included in community C. Since window $W \to \infty$, u belongs in C during the processing of all n edges, and will contribute an increment of 1 to the community degree of each one of its n neighbors for community C. Therefore the sum of increments that u contributes is n. □

Instead of incrementing by 1 the community degrees of all neighbors of a node that have become part of a community, as devised by Eq. (2), we propose here a variation for CoEuS. It employs the network's link structure to improve over the above in-degree-like measure, applying recursively the idea of community participation. We term our variation as *recursive community participation* and we define its value for a node based on the community participation of this node's neighbors, as follows:

$$\mathtt{rcp}(u,t) = \begin{cases} 1, & \text{if } u \in K \\ \dfrac{\sum\limits_{\forall v:(u,v)\in S_t \text{ and } v \in C_{t-1}} \mathtt{cp}(v,t_v)}{|S_t|}, & \text{otherwise} \end{cases} \tag{3}$$

where K is the seed-set for target community C, S_t is the set of elements of the stream S that have arrived at time-step t, C_{t-1} comprises the set of nodes in community C at time-step $t-1$, and t_v is the time-step that edge (u,v) appeared in the stream.

Obviously, our variation employs Eq. (2), which is equal to the fraction of the neighbors of a node that are also members of the community in question. This fraction is an estimation of the probability that a one-step random walk starting from the node will lead to a node that is a member of the community in question. Therefore, the value of Eq. (2) for each node grows with its *involvement* in the community, as we show below with Proposition 2.

Proposition 2. Using our variation, the expected sum of increments some node u contributes to the community degrees of other nodes for a community C, given that window $W \to \infty$, depends on the number of edges incident on u that are processed after u is included in C, and while those nodes are not part of C.

Proof: Assume that there is a set of $m + n$ edges, $m, n \geq 0$, that are incident on u and are processed after u is included in community C. Assume further that for m of these edges, their other endpoint belongs in C when they are being processed, whereas for the rest n their other endpoint does not belong in C when they are being processed. By definition, $\mathtt{degree}_C[v] = 0$ for any node v that does not belong in C. Without loss of generality, we assume that $\frac{\mathtt{degree}_C[v]}{\mathtt{degree}_V[v]} = 1$ for any node v among the m neighbors of u that belong in C. Since window $W \to \infty$, u belongs in C

Algorithm 3: addToCommByEdgeQuality

```
Procedure addToCommByEdgeQuality
    foreach $C \in (I[u] \cup I[v])$ do
        if $u \in C$ then
            $\texttt{degree}_C[v] += \frac{\texttt{degree}_C[u]}{\texttt{degree}_V[u]}$;
        if $v \in C$ then
            $\texttt{degree}_C[u] += \frac{\texttt{degree}_C[v]}{\texttt{degree}_V[v]}$;
        if $u \in C$ then
            $C \leftarrow C \cup \{v\}$;
            $I[v] \leftarrow I[v] \cup C$;
        if $v \in C$ then
            $C \leftarrow C \cup \{u\}$;
            $I[u] \leftarrow I[u] \cup C$;
```

when all $m+n$ edges are being processed and will contribute an increment of $\frac{\texttt{degree}_C[u]}{\texttt{degree}_V[u]}$ to the community degrees of all $(m+n)$ neighbors of u for community C.

Moreover, assume that initially $\frac{\texttt{degree}_C[u]}{\texttt{degree}_V[u]} = \frac{k}{l}$, for $k \leq l$. When one of the m edges is processed, both $\texttt{degree}_C[u]$ and $\texttt{degree}_V[u]$ are increased by 1, whereas when one of the n edges is processed, only $\texttt{degree}_V[u]$ is increased by 1.

Let us denote the random variable that expresses the contribution of u to its i-th neighbor degree by $D_{u,i}$. Then

$$\mathbb{E}(D_{u,i}) = \sum_{j=0}^{i} \binom{i}{j} \frac{\prod_{\rho=1}^{i-j}(m-\rho+1)\cdot\prod_{\rho=1}^{j}(n-\rho+1)}{\prod_{\rho=0}^{i-1}(m+n-\rho)} \frac{k+i-j}{l+i}$$

or, simply,

$$\mathbb{E}(D_{u,i}) = \frac{k}{l+i} + \frac{i\,m}{(m+n)(l+i)}$$

derived by mathematical induction based on the previous equation.

□

When $\mathbb{E}(D_{u,i})$ is high, the probability that a neighbor of u is a member of the community is also high. Thus, using Eq. (2) instead of value 1 to increment the community degree of a node, we *remain vitally focused* in the community.

Algorithm 3 details the above outlined approach, and can replace Lines 15-25 of Algorithm 1. The difference in functionality is in Lines 4 and 6 of Algorithm 3, which increment the participation level of a node in the community using Eq. (2). This results in using Eq. (3) instead of Eq. (2) in Algorithm 3, where we estimate the community participation of each node (Line 4). Our variation favors nodes that are adjacent to well-established members of the community, as such nodes receive a significant increment to their community degree. In contrast, nodes that exhibit low values of Eq. (2) provide insignificant increments to the participation levels of their adjacent nodes. Thus, we prevent irrelevant nodes from shifting the focus of a community.

2.5 Size of the community

CoEuS associates each node included in the expanded community with a community participation value. However, the size of an actual community might be smaller than the one CoEuS examines. Therefore, CoEuS needs to effectively address the issue of determining the size of a community automatically and to remove any irrelevant nodes or less significant nodes.

Algorithm 4: dropTail

```
Procedure dropTail
    $\hat{C} \leftarrow$ reverseSort($C$);
    $\texttt{totalDifference} \leftarrow 0$;
    $\texttt{previous} \leftarrow 0$;
    foreach $v \in \hat{C}$ do
        if $\texttt{previous} > 0$ then
            $\texttt{totalDifference} \leftarrow \texttt{cp}(v) - \texttt{previous}$;
        $\texttt{previous} \leftarrow \texttt{cp}(v)$;
    $\texttt{averageDifference} \leftarrow \frac{\texttt{totalDifference}}{\text{size}(\hat{C})-1}$;
    $\texttt{previous} \leftarrow 0$;
    foreach $v \in \hat{C}$ do
        if $\texttt{previous} > 0$ then
            $\texttt{difference} \leftarrow \texttt{cp}(v) - \texttt{previous}$;
        $\texttt{previous} \leftarrow \texttt{cp}(v)$;
        if $\texttt{difference} < \texttt{averageDifference}$ then
            $\hat{C} \leftarrow \hat{C} \setminus \{v\}$;
        else
            break;
```

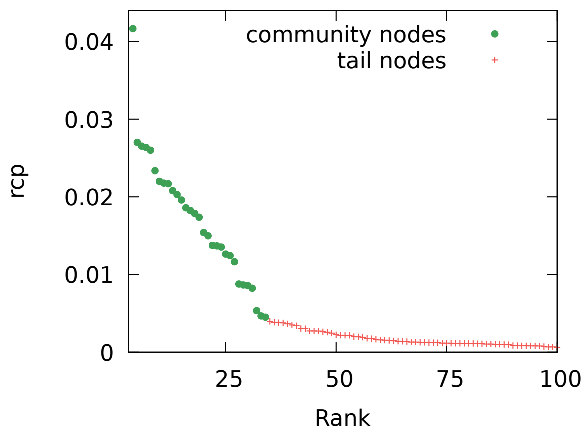

Fig. 3: Ranking of nodes w.r.t. Eq. (3) and the partitioning that Algorithm 4 makes for a community of *dblp*.

Algorithm 4 proposes *dropTail*, a procedure that identifies nodes irrelevant to the community formed with CoEuS and removes them. In this regard, *dropTail* utilizes the community participation values of the nodes included in the community, and allows for fully *automatic*, *on-demand* removal of irrelevant nodes. More specifically, irrelevant nodes exhibit weak ties to the actual community and thus, their respective community participation values are insignificant when compared to the values of other nodes included in the community. This is evident in Figure 3 that illustrates the community participation values of nodes included in a community of a real-world graph, as derived by CoEuS. We observe that ordering nodes according to their community participation values results to a clearly visible *tail*. The distribution of community participation values varies, depending both on the graph and the community in question. Thus, setting a constant *threshold* value and discarding nodes that exhibit lower community participation values to remove such tails is not an option.

Instead, we need to adjust to each particular community and isolate the nodes that belong to the tail through *clustering*. To do so, Algorithm 4 calculates the average distance between two consecutive nodes with regard to their ranking by their associated community participation values (Lines 5-9). Then, we iteratively examine the value distance of two nodes in this ranking, starting from the last node. When this distance is found to be larger than the average distance of nodes, Algorithm 4 stops, as it has spotted a significant gap between the values of two consecutive nodes (Lines 11-18). Figure 3 illustrates the result of this process in the context of an experiment on a real-world network. The average distance between consecutive nodes w.r.t. the ranking by community participation value is 0.00043. The first node from the end that exhibits a gap larger than that from its predecessor is the one ranked 35^{th}. Therefore, Algorithm 4 considers that the tail of irrelevant nodes begins from the 35^{th} node (depicted using red crosses), and the actual community is formed by the first 34 nodes (depicted using green dots).

We note that seed nodes exhibit relatively large community participation values and their inclusion in this process is experimentally found to include more relevant nodes in the tail. Thus, Algorithm 4 does not consider seed nodes.

3 Experimental Evaluation

We proceed by evaluating the performance of CoEuS on a range of networks from various domains. Our experiments measure the impact of the novel techniques of our algorithm and feature comparisons against state-of-the-art community detection approaches that *use the entire graph*. We first discuss the specification of our experimental setting. Then, we evaluate our CoEuS algorithm by answering the following questions:

- What is the impact of employing the edge quality variation of CoEuS with regard to its *accuracy* in detecting communities?
- Can CoEuS automatically determine the *size* of a detected community using our novel *dropTail* clustering procedure?
- Is the accuracy of CoEuS *comparable* to that of state-of-the-art local community detection methods that operate on the *entire graph*?
- What are the benefits of CoEuS with regard to *execution time* as well as *space efficiency* when compared to prior efforts?
- How do the number of communities sought, the network's average degree and the error guarantees of CoEuS impact its effectiveness and efficiency?

3.1 Experimental Setting

Our dataset comprises the 6 publicly available social (*youtube*, *livejournal*, *orkut*, *friendster*), co-authorship (*dblp*), and co-purchasing (*amazon*) networks listed in Table 1[2] and three synthetic networks created using the Lancichinetti-Fortunato-Radicchi (LFR) benchmark [28]. The respective graphs reach up to 1.8 billion edges and possess ground-truth communities which allow for quantifying the accuracy of community detection algorithms. To ensure a fair comparison against earlier approaches, we have adopted

TABLE 1: Graphs of our dataset reaching up to 1.8 billion edges.

Graphs	Nodes	Edges
Amazon	$334,863$	$925,872$
DBLP	$317,080$	$1,049,866$
Youtube	$1,134,890$	$2,987,624$
LiveJournal	$3,997,962$	$34,681,189$
Orkut	$3,072,441$	$117,185,083$
Friendster	$65,608,366$	$1,806,067,135$

the experimental setting of a state-of-the-art algorithm [15] –that also focuses on the networks of Table 1– and use the top-5000 ground-truth communities of each network that possess the highest quality according to [29], after enforcing a minimum community size of 20.

The experiments were carried out on a machine with an Intel® Core™ i5-4590, with a CPU frequency of 3.30GHz, a 6MB L3 cache and a total of 16GB DDR3 1600MHz RAM and the Linux Xubuntu 18.04.3 x86 64 OS. Our implementation, as well as execution tests that enable the reproducibility of our results are publicly available.[3] To maintain node and community degrees, we employ in all our experiments Count-Min sketches. The latter are initialized with the following parameters: $i)$ $d = 7$, and $ii)$ $w = 200{,}000$, so that we obtain 99% confidence that $\epsilon < 10^{-5}$. Our evaluation assumes that 3 random nodes of each ground-truth community are provided to each algorithm as an input seed-set. To measure the accuracy of each algorithm we use the average F1-score achieved for the communities of each graph. All results reported are averages of multiple executions (for various random seed-sets and permutations of the order of edges) and are accompanied with their respective 95% confidence intervals.

3.2 Impact of the Edge Quality Variation

We first examine the behavior of CoEuS when considering our two different techniques of incrementing the community degree of a node. As CoEuS_1, we denote the baseline version of our algorithm, in which the community degree of each node is incremented by 1 for every adjacent node found in the community, and the community participation is estimated using Eq. (2). As CoEuS_{cp}, we designate our proposed approach, in which the community degree of each node is incremented using Eq. (2) and thus, the community participation is estimated using Eq. (3).

We can clearly see in Figure 4 that CoEuS_{cp} achieves an increased F1-score compared to CoEuS_1 for all graphs included in our dataset. Evidently, the edge quality variation we introduce with CoEuS_{cp} heavily impacts the ability of our algorithm to accurately retrieve the members of a community. The improvement for graphs *dblp*, *livejournal*, and *friendster* is particularly *impressive*, increasing from 0.263 to 0.469 (78%), from 0.402 to 0.656 (63%), and from 0.15 to 0.464 (209%), respectively. Significant improvements with regard to F1-score are also achieved for graphs *orkut*, *amazon*, *youtube*.

These results *emphatically verify* that using Eq. (3) we successfully *favor* nodes that are actual members of the community in question, and *penalize* nodes that exhibit weak ties

[2]https://snap.stanford.edu/data/#communities

[3]https://github.com/panagiotisl/CoEuS

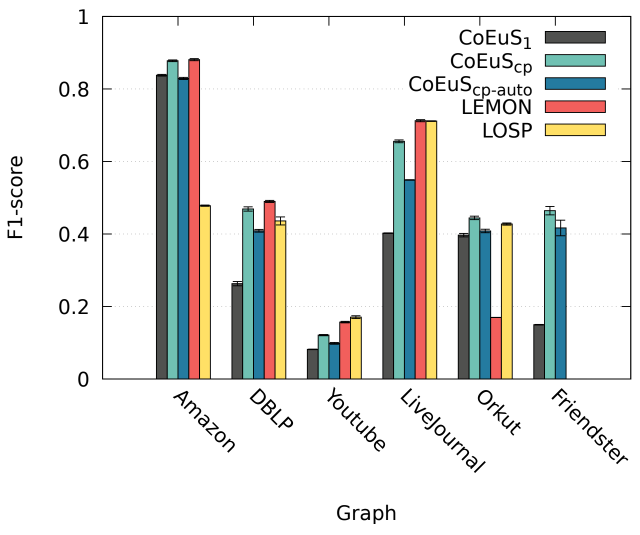

Fig. 4: F1-score comparison for CoEuS$_1$, CoEuS$_{\text{cp}}$, CoEuS$_{\text{cp-auto}}$, LEMON, and LOSP.

with the community, when incrementing their respective community degrees. Thus, the resulting communities are much more accurate than the ones detected when relying entirely on Eq. (2).

We note that this experiment considers as size of each resulting community the size of the respective ground-truth community, for both CoEuS$_1$ and CoEuS$_{\text{cp}}$. We next discuss the evaluation of our automatic size determination clustering algorithm (Algorithm 4), as we cannot assume that the size of a community is known a priori.

3.3 Evaluation of Automatic Size Determination

Community detection via seed-set expansion calls for a stopping criterion for the expanding process. CoEuS employs two techniques to limit the expansion of communities: The first, termed *pruneComm* and detailed in Algorithm 2, is a pruning procedure that is periodically applied to reduce the size of the community. The second, termed *dropTail* and designated in Algorithm 4, is a novel clustering algorithm that is applied on the resulting community of CoEuS to separate the nodes that exhibit weak ties with the community and should be removed. In this experiment, we evaluate the effectiveness of our clustering algorithm by comparing the average F1-score of CoEuS$_{\text{cp}}$ and CoEuS$_{\text{cp-auto}}$; for CoEuS$_{\text{cp}}$ we assume that the size of each community is known a priori, whereas CoEuS$_{\text{cp-auto}}$ automatically derives the size of a community using Algorithm 4.

We can see in Figure 4 that our CoEuS$_{\text{cp-auto}}$ offers impressive performance, as the difference with the F1-score of CoEuS$_{\text{cp}}$ is in most cases negligible. More specifically, the difference in F1-score is under 0.1 for all networks of our dataset and 0.05 on average. This result strongly highlights the effectiveness of Algorithm 4 to determine the *size of a community automatically*. We also note that Algorithm 4 is extremely efficient, both time- and space-wise, requiring only two passes over each resulting community (about 100 nodes), without any access to the graph's elements. In contrast, other size determination techniques such as the ones employed in [15], [16] necessitate calculations of complex community quality measures like that of Eq. (1) for every possible size of each community and require the presence of the *entire graph*.

3.4 Comparison against state-of-the-art non-streaming local community detection algorithms

We now proceed with comparing our graph stream algorithm against state-of-the-art non-streaming local community detection algorithms. Our comparison focuses on LEMON[4] [15] and LOSP[5] [16] as they are shown to outperform all prior approaches [10], [13], [30], whereas the more recent SCODA [21] reports significantly lower F1-scores and does not allow overlaps.

3.4.1 F1-score comparison

We begin the comparison of CoEuS$_{\text{cp-auto}}$ with earlier non-streaming approaches as far as their accuracy on detecting communities is concerned. We initialize LEMON and LOSP with three random seeds of each ground-truth community of our dataset and report averages of multiple executions. Note, that we were unable to retrieve results for *friendster* with either of the two algorithms due to memory requirements. Regarding LEMON, this is also the case for *orkut*; however, we include here the results reported for this network in [15].

As we can see in Figure 4, our algorithm is extremely competitive w.r.t. accuracy, despite the fact that it operates on a graph stream setting using partial knowledge. LEMON slightly outperforms CoEuS$_{\text{cp-auto}}$ for the four smaller graphs of our dataset, with an average F1-score difference of 0.089. Regarding *orkut*, CoEuS$_{\text{cp-auto}}$ is far more accurate achieving an F1-score of 0.408 against LEMON's 0.17. Moreover, LOSP is slightly better than CoEuS$_{\text{cp-auto}}$ for *amazon*, *youtube*, *livejournal*, and *orkut* but much worse for *dblp*. Finally, our algorithm is able to achieve the noticeable F1-score of 0.417 for the largest graph of our dataset, which both LEMON and LOSP fail to handle due to its size.

These results are particularly impressive as the graph stream setting of CoEuS, is much more restrictive than the setting LEMON, LOSP and other prior seed-set expansion methods operate on. CoEuS processes each edge of the graph as it becomes available and maintains rather limited information for each node and community. Hence, it is indeed surprising that our algorithm achieves *accuracy comparable to or better than* methods that utilize the entire graph structure. Furthermore, it is evident in Figure 4 that our effective novel graph stream techniques enable CoEuS to easily scale to large graphs, which other community detection methods fail to handle.

3.4.2 Execution time and space efficiency comparison

Having shown that CoEuS$_{\text{cp-auto}}$ is competitive or better than state-of-the-art non-streaming algorithms as far as accuracy is concerned, we report results concerning *execution time* and *space* efficiency in this section.

Table 2 illustrates a comparison on the execution time between CoEuS$_{\text{cp-auto}}$, LEMON and LOSP. Regarding

[4]https://github.com/YixuanLi/LEMON
[5]https://github.com/PanShi2016/LOSP_Plus

TABLE 2: Execution time comparison.

Graphs	**CoEuS$_{\text{cp-auto}}$**	**LEMON**	**LOSP**
Amazon	0.01 sec	3.12 sec	1.233 sec
DBLP	0.011 sec	7.276 sec	4.828 sec
Youtube	0.085 sec	11.383 sec	9.189 sec
LiveJournal	0.543 sec	28.14 sec	45.291 sec
Orkut	2.076 sec	–	60.8 sec
Friendster	31.449 sec	–	–

CoEuS$_{\text{cp-auto}}$, we consider a streaming setting in which we process every edge of each graph to update our structures and expand the communities in question. For LEMON and LOSP we must read the entire graph in-memory, as only then can we sequentially expand each community in question. For our experimentation, we use only one CPU core as parallel execution is not an option for LEMON and LOSP, whose code is written in Python and MATLAB, respectively. The results capture the average time needed for one community of each network. Table 2 demonstrates our algorithm is extremely fast. In particular, CoEuS$_{\text{cp-auto}}$ scales to networks reaching billions of edges requiring only an impressive average of 31.449 seconds for a community of the *friendster* network. In contrast, LEMON and LOSP need equivalent time to expand the communities of much smaller networks.

Even though these results clearly show that CoEuS is considerably faster than prior approaches, they are not indicative of CoEuS's speed in a real streaming setting. CoEuS is able to return the communities in question *on-demand* as we process the stream in *real-time*. The measurements reported in Table 2 additionally consider edge processing. In a real-life setting, we expect this processing to execute faster than edges are made available. In this regard, the actual response time of CoEuS in a streaming setting is *in the order of milliseconds*, regardless of the size of the graph. Yet, the results of Table 2 indicate that CoEuS is a very attractive option even for non-streaming settings.

The space requirements of CoEuS depend only on the desired approximation quality of the two Count-Min sketches we employ and the number of communities in question. The two sketches are configured to use in our experiments the same space regardless of the graph. Thus, the space requirements of CoEuS are independent of the size of the graph. In contrast, LEMON and LOSP need to maintain in-memory the entire graph structure. The additional space required by all three algorithms to hold the communities we seek is linear to the number of the communities and fairly insignificant compared to the graph structure.

Our measurements verify that the total space requirements of CoEuS are remarkably low. The two sketches we employ require just 21.36MB, and handle appropriately even the largest graph of our dataset, reaching up to 1.8 billion edges. In contrast, the space requirements of LEMON and LOSP grow with the number of edges of a graph. The largest graph we are able to handle with LEMON is *livejournal* with 34 million edges for which more than 2.5GB are needed. Similarly, LOSP needs more 8GB to process *orkut* with 117 million edges. Both algorithms result in memory errors when processing larger graphs.

3.4.3 Impact of window W

One of the parameters of CoEuS is the size of a window W, which determines how often the communities sought are pruned. This step contributes to the efficiency of CoEuS as allowing communities to grow uncontrollably would cause significant processing overhead. In our experiments, W is set to $10{,}000$ edges as we have found that using larger values might slow down CoEuS moderately. More specifically, setting $W = 15{,}000$ causes on average 4.7% execution time increase. Setting $W = 5{,}000$, also increases execution time by 5.1% on average, due to the pruning step which is executed twice as often. Note here, that this choice of W corresponds to a maximum community size $s = 100$. If one opts to use a different value of s, the choice of W should be reconsidered.

Regarding accuracy, we have found the impact of W to be negligible and no value would consistently outperform setting $W = 10{,}000$ for the graphs of our dataset.

3.5 Experiments on Synthetic Networks

Our next set of experiments focuses on synthetic graphs. We first investigate how the number of targeted communities impacts the processing time of CoEuS$_{\text{cp-auto}}$. Then, we discuss the respective impact caused by a network's average degree. Finally, we evaluate the effectiveness of CoEuS$_{\text{cp-auto}}$ with regards to the error guarantees provided by the Count-Min sketches employed by the algorithm.

We create 3 synthetic networks of 1 million nodes each, using the Lancichinetti-Fortunato-Radicchi (LFR) benchmark [28].[6] The 3 networks are generated to exhibit an average degree of 10, 20, and 30, respectively. For each synthetic network we randomly select $4{,}000$ of the generated communities to use in this set of experiments.

3.5.1 Impact of the number of communities

Figure 5 illustrates the processing time–per–edge of CoEuS$_{\text{cp-auto}}$ when varying both the number of communities we seek and the average degree of the network. We observe that for a given average degree of the network, increasing the number of communities we seek, results in increased processing time–per–edge. That is, CoEuS$_{\text{cp-auto}}$ scales almost linearly with the number of communities sought, as we vary this number from $1{,}000$ to $4{,}000$.

3.5.2 Impact of the network's average degree

Figure 5 also shows that for a given number of communities we wish to uncover, the processing time of CoEuS$_{\text{cp-auto}}$ remains relatively stable as we vary the network's average degree. In particular, the processing time–per–edge of CoEuS$_{\text{cp-auto}}$ ranges from 8.28 to 9.32 μs when seeking $1{,}000$ communities, from 17.35 to 19.54 μs when seeking $2{,}000$ communities, from 25.13 to 27.54 μs when seeking $3{,}000$ communities, and from 36.01 to 40.7 μs when seeking $4{,}000$ communities. The slight increase reported when increasing the network's average degree is attributed to the fact that the communities sought expand quicker as the network's average degree grows. However, due to the pruning step of CoEuS$_{\text{cp-auto}}$ this impact becomes negligible.

[6] We use a mixing parameter of 0.1, a maximum degree of 100, and communities with $20 - 100$ member nodes.

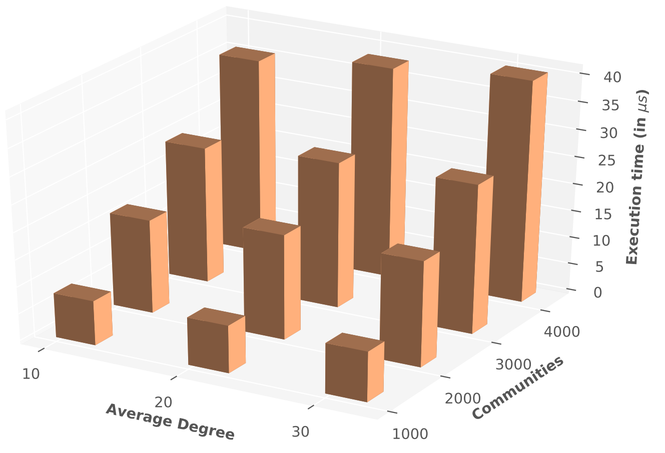

Fig. 5: Impact of average degree and number of communities on the average processing time per edge (in μs).

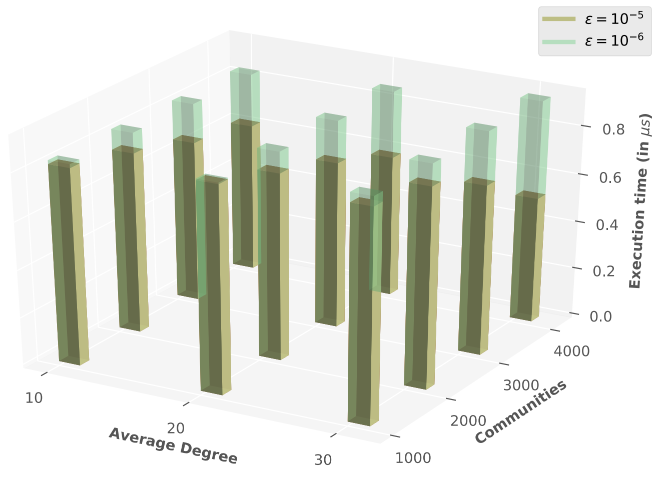

Fig. 6: Impact of COUNT-MIN sketches' error guarantees on F1-score, when varying the network's average degree and the number of communities sought.

3.5.3 Impact of error guarantees

We further investigate the performance of COEUS$_{\text{cp-auto}}$ regarding the effectiveness of the algorithm on detecting the communities of synthetic networks. Figure 6 illustrates the F1-score that COEUS$_{\text{cp-auto}}$ achieves when varying the network's average degree and the number of communities sought. We examine two variations of COEUS$_{\text{cp-auto}}$ regarding the COUNT-MIN sketches employed: i) a variation in which the sketches provide 99% confidence that $\epsilon < 10^{-5}$, and ii) a variation in which the sketches provide 99% confidence that $\epsilon < 10^{-6}$. We note that $\epsilon < 10^{-5}$ requires 23MB of memory, whereas $\epsilon < 10^{-6}$ needs 214MB of memory.

We observe in Figure 6 that both the number of communities sought and the network's average degree have an effect on the accuracy of COEUS$_{\text{cp-auto}}$ in detecting communities. We first discuss the case of varying the number of communities that appears to be more impactful. Using sketches that provide 99% confidence that $\epsilon < 10^{-5}$, and for an average degree of 10, when the number of communities grows from $1{,}000$ to $4{,}000$, the F1-score drops from 0.823 to 0.621. Similarly, for average degrees of 20 and 30, the F1-score drops from 0.867 to 0.592 and from 0.891 to 0.528, respectively. On the contrary, when the sketches of COEUS$_{\text{cp-auto}}$ provide 99% confidence that $\epsilon < 10^{-6}$, the F1-score remains very stable, i.e., the standard deviation of the F1-scores for a varying number of communities is below 0.003 for all cases of average degree we examine. Clearly, using sketches with $\epsilon < 10^{-5}$ is not effective when searching more than $1{,}000$ communities in these networks. However, COEUS$_{\text{cp-auto}}$ can be easily set to use sketches that provide better error guarantees and maintain the same effectiveness as the number of communities grows.

The performance of COEUS$_{\text{cp-auto}}$ also differentiates when varying the network's average degree. However, this can be attributed to the topology and community structure of the 3 different networks we examine. It is worth mentioning here that COEUS$_{\text{cp-auto}}$ is very competitive against `LEMON` as far as synthetic networks are concerned as well. `LEMON` performs better for the two smaller networks – with average F1-scores of 0.94 and 0.93 against 0.83 and 0.87– whereas COEUS$_{\text{cp-auto}}$ is better for the largest synthetic network – with average F1-score 0.93 against 0.88.

4 RELATED WORK

Our work lies in the intersection of community detection in stream graphs and local community detection via seed-set expansion. Below, we outline relevant research efforts.

Local community detection via seed-set expansion: Numerous recent approaches depart from the direction of working with the entire graph structure. Instead, they focus on detecting *local* communities in time functional to the size of the community and are thus able to support large scale graphs. Such approaches usually operate using a seed-set of nodes which they expand to a community. Kloster and Gleich [13] propose a deterministic local algorithm to compute heat kernel diffusion and study the communities it produces when initiated with seed nodes. `LEMON` [15] also uses seeds to perform short random walks and forms an approximate invariant subspace termed *local spectra*. Then, `LEMON` looks for the minimum 1-norm vector in the span of this *local spectra* such that the seeds are in its support. To determine the size of the community, the authors of [15] measure the conductance of the community as they increase its size, and stop at the first relative minimum encountered. `LOSP` [16] samples locally in the graph to get a small subgraph containing most of the latent members. Then, a family of local spectral approximation methods are used to extract the local community from the sampled subgraph via a Krylov subspace formed by short-random-walk diffusion vectors. All three above approaches are similar to our setting as they expand a seed-set of nodes into a community. However, none of them is appropriate for graph streams. `LDLC` [14] focuses on egonets of nodes in networks and performs hierarchical link clustering to detect *all* the overlapping communities of a node. In addition, `LDLC` uses a measure of *dispersion* to detect nodes that share multiple communities and thus avoids to group together overlapping parts of communities. Our COEUS is different as it uses a seed-set of nodes and expands it into a single community.

Streaming community detection: Yun et al. [22] consider settings in which the size of the network is so large that maintaining the respective graph is prohibitive. Thus, they study the problem of clustering the nodes of a graph to communities in a streaming setting where rows of the adjacency matrix of the graph are revealed sequentially. They propose an online algorithm with space complexity that grows sub-linearly with the size of the network. Our streaming setting does not assume that rows of the adjacency matrix are completely revealed to us. Instead, we consider that edges involving any node of the graph may arrive at any moment. Moreover, we are unaware of the size of the graph, which grows with time. Zakrzewska and Bader [31] propose a dynamic seed set expansion algorithm for community detection. In particular, they consider that edges may be inserted to or removed from the graph dynamically and detect the local community of a seed set by incrementally adjusting to the changes of the graph. The latter allows for faster execution compared to an algorithm that requires re-computation after every update at the cost of slightly worse community quality. Our approach is different as we assume that we cannot maintain the entire graph in-memory, whereas the incremental adjustments that [31] performs do impose such a requirement. Moreover, we suggest a significantly more cost-effective recomputation of the local community at every step. Hollocou et al. [21] consider an edge streaming setting and assign all the nodes of a graph to non overlapping communities using only two integers per node that hold: i) the node's degree, and ii) the current community index assigned to the node. Their work is heavily based on the observation that if one picks uniformly at random an edge of the graph, this edge is more likely to link nodes of the same community, than nodes from distinct communities. This is expected to be true as nodes tend to be more connected within a community than across communities, thus, if we process edges in a random order we expect many intra-community edges to arrive before the inter-community edges. However, this requires that we already hold the graph in its entirety and we are able to select its edges one by one uniformly at random. We operate on a more practical assumption that the edges of the graph arrive at *no particular order*.

Some earlier approaches focus on dynamic community detection in graph streams. Wang et al. [32] transform a content-based network into a multi-mode network, termed NEI network. The evolution of communities is captured by performing heterogeneous random walks in the NEI network. Our approach is different as we consider that maintaining the entire graph structure is prohibitive. Lai et al. [33] propose the use of top-k neighbor and candidate lists to maintain only the most useful information of an evolving graph. Our data stream model setting is different as [33] considers the use of an *offline* component that uses these candidate lists to form communities. Duan et al. [34] present an incremental algorithm to update the partition of a graph segment when adding a new arriving graph. Community detection is achieved through random walks with restart, and requires maintaining the entire graph structure. Finally, [35] maintains at each given moment the current graph of interactions in main memory, and store previous graphs of interactions on disk. This is different than our data stream model setting which considers single pass access and limited working memory.

A preliminary version of our work appeared in [36]. Here, we formulate and prove Propositions 1 and 2 that justify the improved performance of our variation on Algorithm 3. Additionally, we improve Algorithm 1 by introducing an additional index that helps improve our efficiency. Lastly, we further evaluate CoEuS by creating synthetic networks and investigating the impact of the number of communities sought, the network's average degree, and CoEuS's sketches error ϵ.

5 Conclusions and Future Directions

In this work, we propose, develop and experiment with CoEuS, a novel graph stream community detection algorithm that expands seed-sets of nodes into communities. To the best of our knowledge CoEuS is the first streaming algorithm that performs community detection using space sublinear to the number of edges without imposing any restrictions in the order in which edges arrive in the stream. CoEuS processes a stream of edges and maintains limited information about the respective graph, concerning the nodes' degrees, the participation of nodes into communities and the nodes that comprise each community we seek. In addition, we propose two methods that enhance the effectiveness of our approach significantly. The first method places emphasis on the quality of an edge w.r.t. a community and is able to better preserve the focus of a community as the latter is expanding. The second technique allows for automatic, on-demand determination of the size of a community through a novel clustering technique, tailored to the needs of CoEuS.

We compare CoEuS with a non-streaming local community detection method that reportedly outperforms other recent approaches. Using large-scale real-world networks from various domains, as well as synthetic networks, we show that CoEuS offers accuracy that is equivalent to or better than that of methods exploiting the entire graph. This holds true even though our approach operates on a graph stream. The two additional methods we propose in this paper, contribute to the effectiveness and efficiency of CoEuS, by improving its accuracy and allowing for *real-time* determination of the size of each community. Furthermore, we examine the requirements of CoEuS and show that our algorithm is clearly superior than prior approaches with regard to both execution time and space used. Our CoEuS algorithm proves to be not only an extremely accurate graph stream algorithm, but a very attractive option for large-scale community detection in general.

A direction we wish to explore in the future is handling graph streams that represent weighted networks. Furthermore, we will investigate and develop variations of CoEuS that will allow for processing both insertions and removals of edges in the graph.

References

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

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

[3] 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.

[4] A. Clauset, M. E. Newman, and C. Moore, “Finding community structure in very large networks,” *Physical review E*, vol. 70, no. 6, p. 066111, 2004.

[5] M. E. J. Newman and M. Girvan, “Finding and evaluating community structure in networks,” *Phys. Rev. E*, vol. 69, no. 2, p. 026113, Feb. 2004.

[6] P. Pons and M. Latapy, “Computing communities in large networks using random walks,” in *Proc. of the 20th Computer and Information Sciences Int. Symp.*, 2005, pp. 284–293.

[7] Y.-Y. Ahn, J. P. Bagrow, and S. Lehmann, “Link communities reveal multiscale complexity in networks,” *Nature*, vol. 466, no. 7307, pp. 761–764, 2010.

[8] T. Evans and R. Lambiotte, “Line graphs, link partitions, and overlapping communities,” *Physical Review E*, vol. 80, p. 016105, 2009.

[9] D. F. Gleich and C. Seshadhri, “Vertex neighborhoods, low conductance cuts, and good seeds for local community methods,” in *Proc. of the 18th ACM SIGKDD*, 2012, pp. 597–605.

[10] J. J. Whang, D. F. Gleich, and I. S. Dhillon, “Overlapping community detection using neighborhood-inflated seed expansion,” *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 5, pp. 1272–1284, 2016.

[11] J. Yang and J. Leskovec, “Community-affiliation graph model for overlapping network community detection,” in *Proc. of the 12th IEEE ICDM*, 2012, pp. 1170–1175.

[12] ——, “Overlapping community detection at scale: a nonnegative matrix factorization approach,” in *Proc. of the 6th ACM WSDM*, 2013, pp. 587–596.

[13] K. Kloster and D. F. Gleich, “Heat kernel based community detection,” in *The 20th ACM SIGKDD Int. Conf.*, 2014, pp. 1386–1395.

[14] P. Liakos, A. Ntoulas, and A. Delis, “Scalable link community detection: A local dispersion-aware approach,” in *Proc. of the 2016 IEEE Int. Conf. on BigData, Washington DC*, pp. 716–725.

[15] Y. Li, K. He, K. Kloster, D. Bindel, and J. E. Hopcroft, “Local spectral clustering for overlapping community detection,” *ACM Trans. on Know. Disc. from Data*, vol. 12, no. 2, pp. 17:1–17:27, 2018.

[16] K. He, P. Shi, D. Bindel, and J. E. Hopcroft, “Krylov subspace approximation for local community detection in large networks,” *ACM Trans. on Know. Disc. from Data*, vol. 13, no. 5, pp. 52:1–52:30, 2019.

[17] P. Liakos, K. Papakonstantinopoulou, and A. Delis, “Realizing memory-optimized distributed graph processing,” *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 4, pp. 743–756, 2018.

[18] A. McGregor, “Graph stream algorithms: a survey,” *SIGMOD Record*, vol. 43, no. 1, pp. 9–20, 2014.

[19] S. Guha, A. McGregor, and D. Tench, “Vertex and hyperedge connectivity in dynamic graph streams,” in *Proc. of the 34th ACM PODS, Melbourne, Australia*, 2015, pp. 241–247.

[20] Z. Abbas, V. Kalavri, P. Carbone, and V. Vlassov, “Streaming graph partitioning: An experimental study,” *PVLDB*, vol. 11, no. 11, pp. 1590–1603, 2018.

[21] A. Hollocou, J. Maudet, T. Bonald, and M. Lelarge, “A linear streaming algorithm for community detection in very large networks,” *CoRR*, 2017. [Online]. Available: http://arxiv.org/abs/1703.02955

[22] S. Yun, M. Lelarge, and A. Proutière, “Streaming, memory limited algorithms for community detection,” in *Proc. of the 28th NIPS, Montreal, Canada*, 2014, pp. 3167–3175.

[23] M. W. Mahoney, L. Orecchia, and N. K. Vishnoi, “A local spectral method for graphs: With applications to improving graph partitions and exploring data graphs locally,” *J. Mach. Learn. Res.*, vol. 13, no. 1, pp. 2339–2365, Aug. 2012.

[24] S. Galhotra, A. Bagchi, S. Bedathur, M. Ramanath, and V. Jain, “Tracking the conductance of rapidly evolving topic-subgraphs,” *PVLDB*, vol. 8, no. 13, pp. 2170–2181, 2015.

[25] G. Cormode and S. Muthukrishnan, “An improved data stream summary: the count-min sketch and its applications,” *J. Algorithms*, vol. 55, no. 1, pp. 58–75, 2005.

[26] J. Leskovec, J. M. Kleinberg, and C. Faloutsos, “Graphs over time: densification laws, shrinking diameters and possible explanations,” in *Proc. of the 11th ACM SIGKDD*, 2005, pp. 177–187.

[27] J. Yang and J. Leskovec, “Structure and overlaps of ground-truth communities in networks,” *ACM Trans. on Intelligent Systems and Technology*, vol. 5, no. 2, p. 26, 2014.

[28] A. Lancichinetti, S. Fortunato, and F. Radicchi, “Benchmark graphs for testing community detection algorithms,” *Physical review E*, vol. 78, no. 4, p. 046110, 2008.

[29] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney, “Statistical properties of community structure in large social and information networks,” in *Proc. of the 17th WWW*, 2008, pp. 695–704.

[30] I. M. Kloumann and J. M. Kleinberg, “Community membership identification from small seed sets,” in *The 20th ACM SIGKDD*, 2014, pp. 1366–1375.

[31] A. Zakrzewska and D. A. Bader, “A dynamic algorithm for local community detection in graphs,” in *Proc. of the 2015 IEEE/ACM ASONAM*, pp. 559–564.

[32] C. Wang, J. Lai, and P. S. Yu, “Neiwalk: Community discovery in dynamic content-based networks,” *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 7, pp. 1734–1748, 2014.

[33] J. Lai, C. Wang, and P. S. Yu, “Dynamic community detection in weighted graph streams,” in *Proc. of the 13th SIAM Int. Conf. on Data Mining, Austin, TX*, 2013, pp. 151–161.

[34] D. Duan, Y. Li, Y. Jin, and Z. Lu, “Community mining on dynamic weighted directed graphs,” in *Proc. of the 1st ACM Int. Work. on Complex Networks Meet Information & Knowledge Management, Hong Kong, PRC*, 2009, pp. 11–18.

[35] C. C. Aggarwal and P. S. Yu, “Online analysis of community evolution in data streams,” in *Proc. of the 2005 SIAM Int. Conf. on Data Mining, Newport Beach, CA*, pp. 56–67.

[36] P. Liakos, A. Ntoulas, and A. Delis, “COEUS: community detection via seed-set expansion on graph streams,” in *Proc. of the 2017 IEEE Int. Conf. on Big Data, Boston, MA*, 2017, pp. 676–685.

Panagiotis Liakos is a Postdoctoral researcher at the University of Athens, where he obtained his Ph.D. on distributed and streaming graph processing techniques. His research interests include graph mining and information retrieval, with a particular focus on mining large scale graphs and streams of social activity.

Katia Papakonstantinopoulou is a Lecturer of Computer Science at the Department of Informatics at the Athens University of Economics and Business. She holds B.Sc., M.Sc., and Ph.D. degrees from the University of Athens. Her research interests are in Algorithmic Game Theory and Social and Information Network Analysis.

Alexandros Ntoulas is an Assistant Professor of Computer Science at the University of Athens. He holds both a Ph.D. and an M.Sc. in Computer Science from the University of California, Los Angeles and a Diploma in Computer Engineering from the University of Patras. His reaserch interests are in WWW and Big Data. He has received a best paper award (ICDE 2005), a best paper runner-up award (WWW 2009), and a best applied data science reviewer award (KDD 2017).

Alex Delis is a Professor of Computer Science at the University of Athens. His research interests are in Distributed and Virtualized Data Systems. He holds both a Ph.D. and an M.Sc. in Computer Science from the University of Maryland at College Park and a Diploma in Computer Engineering from the University of Patras. He is a member of IEEE Computer Society, ACM, and the Technical Chamber of Greece.