CoEuS: Community Detection via Seed-set Expansion on Graph Streams

Panagiotis Liakos*, Alexandros Ntoulas†‡ and Alex Delis*§
*University of Athens, Athens, Greece, Email: {p.liakos, ad}@di.uoa.gr
†LinkedIn, Mountain View, CA, Email: ntoulas@gmail.com
§New York University Abu Dhabi, Abu Dhabi, U.A.E.

***Abstract*—We examine the problem of effective identification of community structure of a network whose elements and their respective relationships manifest through streams. The problem has recently garnered much interest as it appears in emerging computational environments and concerns critical applications in diverse areas including social computing, web analysis, IoT and biology. Despite the already expended research efforts in detecting communities in networks, the unprecedented volume that real-world networks now reach, renders the task of revealing community structures extremely burdensome. The sheer size of such networks oftentimes makes their representation in main memory impossible. Thus, processing the developing graphs to extract the underlying communities remains an open challenge. In this paper, we propose a graph-stream community detection algorithm that expands seed-sets of nodes to communities. We consider a stream of edges and aim at processing them to form communities without maintaining the entire graph structure. Instead, we maintain very limited information regarding the nodes of the graph and the communities we seek. In addition to our novel streaming approach, we both develop a technique that increases the accuracy of our algorithm considerably and propose a new clustering algorithm that allows for automatically deriving the size of the communities we seek to detect. Our experimental evaluation using ground-truth communities for a wide range of large real-word networks shows that our proposed approach does achieve accuracy comparable or even better to that of state-of-the-art non-streaming community detection algorithms. More importantly, the attained improvements in both execution time and memory space requirements are remarkable.**

I. Introduction

Graph structures attract significant attention as they allow for representing entities of various domains as well as the relationships these entities entail. Real-world networks are commonly portrayed using graphs and are often massive. Despite their size, such networks exhibit a high level of order and organization, a property frequently referred to as community structure [8]. 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 Twitter and Facebook. Identifying the social communities of an individual enables us to perform recommendations for new connections. Moreover, by better understanding the membership of an individual to various organizational groups, we can provide more informative and engaging social network feeds. In addition to social networks, community detection is successfully applied to numerous other types of networks, such as biological or citation networks. In the former, we are particularly interested in inferring communities of interacting proteins, whereas in the latter we wish to uncover relationships between disciplines or the citation patterns of authors [7].

In the last two decades a plethora of community detection methods has been proposed. Initially, the focus has been on non-overlapping communities [2], [4], [21], [22]. More recent approaches, however, allow for nodes to belong to more than one community [1], [6], [9], [23], [24], [25]. Still, these approaches are not applicable to the massive graphs of the Big Data era, as they focus on the *entire* graph structure and do not scale with regards to both execution time and memory consumption. Recent efforts manage to scale as far as execution time is concerned by focusing on the local structure and expanding exemplary seeds-sets into communities [12], [16], [10], [17]. Such a seed-set expansion setting can be applied to numerous real world applications, e.g., given a few researchers focusing on Big Data 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 [18].

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 [20]. This is a new direction in the field of community detection and to the best of our knowledge no prior approach has considered such a setting without imposing restrictions on the order in which edges are made available [11], [27]. In this paper, we propose CoEuS, a novel 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 idea for community detection in graph streams,

This work has been partially supported by the University of Athens Special Account of Research Grants № 13233.
‡This work was done before author joined LinkedIn.

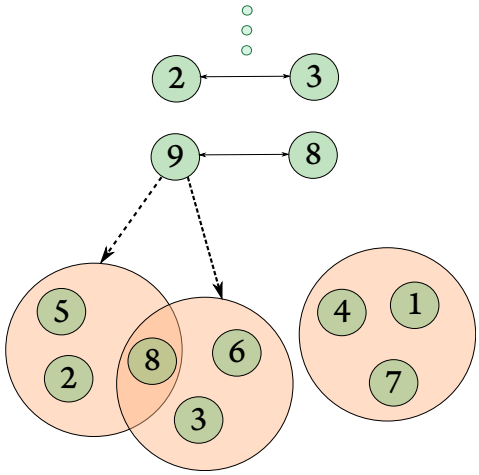

Fig. 1: A stream comprising the edges of an undirected graph and a set of communities initialized with a few seed nodes. For every edge of the stream we wish to evaluate whether the adjacent nodes belong to the communities we examine.

we propose two algorithms to enhance the effectiveness of CoEuS. The first one focuses on better quantifying the quality of each edge w.r.t. to a community. The second one 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 on various large scale real-world graphs show that CoEuS is extremely competitive with regard to *accuracy* against approaches that employ the entire graph structure and 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. Moreover, CoEuS is able to derive the communities in question inordinately faster. For instance we show that CoEuS is more than 17 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 as we process the graph stream. This is particularly important, as even if we could afford to use space linear to the number of a graph's edges, no other approach is able to update communities as new edges arrive with no additional *significant* computational cost.

In summary, we make the following contributions:

- We propose CoEuS, a novel community detection algorithm that can operate 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 it is extremely competitive with prior approaches that cannot operate on graph streams and require the presence of the entire graph structure. In addition, we show that both the execution time and space requirements of CoEuS are astonishingly low.

Our paper is organized as follows: We first formulate our problem and discuss our approach for graph stream community detection in Section II. In Section III, we extensively evaluate our approach and its variations with regard to accuracy, execution time, and space requirements. Section IV reviews related work and finally, Section V concludes our paper.

II. Community Detection via Seed-Set Expansion on Graph Streams

This section first formulates the problem we target 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.

A. Problem formulation

Consider a streaming sequence of unique unordered pairs $e = \{u, v\}$ where $u, v \in V$. 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 the set of vertices $\{v_1, v_2, \ldots, v_n\}$ and E is the set of undirected edges $\{e_1, e_2, \ldots, e_m\}$. Given a community seed-set $K = \{k_1, k_2, \ldots, k_l\} \in V$, our goal is to extend it to a community C. Figure 1 shows such a graph stream with two visible arriving edges, and three seed-sets that are to be extended to communities.

A community is generally thought to be a set of nodes of a graph that are tightly connected to each other and maintain very few ties with the rest of the graph's nodes [21]. 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 [9], [16], [19], [23] quality function in the field of community detection is the *conductance* of a community. More specifically, conductance $\phi(C)$ of a community C is formally defined as:

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

where:

$$adj(C_i, C_j) = |\{(u, v) \in E : u \in C_i, v \in C_j\}|.$$

Several methods try 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 of the community. However, tracking the conductance of all possible communities as we process the edges of stream is inefficient with regard to both time and space. Instead, we introduce here *community participation* $cp(u)$ of a node u in a community, that measures a node's u participation level in a community. In particular the community participation of node u in community C is defined as:

$$cp(u) = \frac{|\{(u, v) \in E : v \in C\}|}{|\{(u, v) \in E\}|}, \tag{2}$$

i.e., community participation of a node in a community is the fraction of its adjacent nodes in the graph that are part of the community. Our intuition is that including nodes exhibiting high values of *cp* to a community C will result to a low value of *conductance* for the community. To this end, our approach employs 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 algorithms.

B. Space complexity

The motivation behind graph stream algorithms lies in the fact that many real-world networks nowadays reach sizes that are simply too large. Thus, graph algorithms are unable to store and process the respective graphs in their entirety [20]. In contrast, graph stream algorithms process a stream comprising the edges of the graph in the order in which these edges arrive over time using *limited memory*.

Earlier streaming community detection approaches have successfully revealed the community structure of graphs streams with limited memory requirements. However, the latter were minimized at the expense of additional constraints on the order in which the edges of the stream arrive. In particular, Yun et al. [27] consider a data stream model in which rows of the adjacency matrix of the graph are revealed sequentially. In such a setting we are aware at any moment of all neighbors of certain nodes. Thus, we can apply community detection with partial information on the subgraphs as they are revealed to us. Memory requirements are kept low as we can discard at each step all information that was made available in earlier steps. Moreover, `SCoDa` [11] 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. The latter assumption is not true for graph streams.

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 [11], [27]. We focus instead on estimating the participation level of each node of the graph in each of the communities we examine, according to Eq. (2). In this context, we need to keep track of the following aspects as we process a graph stream:

1) **degrees**: the total number of nodes each node in the graph is adjacent to, i.e., the degree of each node in the graph,
2) **community degrees**: the degree of each node in the subgraph of each community, and
3) **communities**: the nodes that comprise each community we examine.

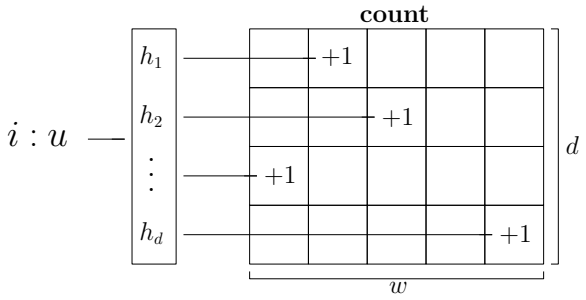

Fig. 2: COUNT-MIN Sketch update process.

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 ($|C'|+1$ integers for each node of the graph). More specifically, we need one integer for the degree of each node in the graph, and one integer for each community we examine to hold the degree of the node in the subgraph that comprises the nodes of the community. Given that the number of communities we examine can be large we decided to use COUNT-MIN sketches to hold the $|C'|+1$ integers.

The COUNT-MIN sketch [5] 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 two-dimensional array of w columns and d rows, where $w = \lceil \frac{e}{\epsilon} \rceil$, $d = \lceil \frac{ln(1)}{\delta} \rceil$, 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 in 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 *id* using the indices of the node and the community and create an update $(i : u, 1)$, indicating that the count of $i : u$ should be incremented by 1. The array $count$ is updated as follows: for each row j of $count$ we apply the corresponding hash function to obtain a column index $k = h_j(i : u)$ and increment the value in row j, column k of the array by 1, i.e., $count[j, k] += 1$. This allows us to retrieve at any time an (over)estimation of the count of an event $i : u$ using the least value in the array for $i : u$, i.e., $\hat{a_{i:u}} = min_j count[j, h_j(i : u)]$.

C. Our CoEuS Algorithm for Streaming Community Detection

In this section, we discuss the details of our algorithm for streaming community detection, termed COEUS.[1] The pseudocode of COEUS is given in Algorithm 1.

Input/Output: COEUS takes as its input two parameters: The first one is a set of community seed-sets $K' = \{K_1, K_2, ..., K_s\}$, where each $K_i = \{k_1, k_2, \ldots, k_l\} \in V$.

[1] In Greek mythology Coeus was the Titan of intellect, the axis of heaven around which the constellations revolved and probably of heavenly oracles.

Algorithm 1: CoEuS(S, K')

```
   input : A set of community seed-sets K',
           and a graph stream S.
   output: A set of communities C'.
1  begin
2      foreach K ∈ K' do
3          C ← {};
4          foreach k ∈ K do
5              C[k] = 1;
6          C'.put(C);
7      while ∃(u, v) ∈ S do
8          degree_V[u] += 1;
9          degree_V[v] += 1;
10         foreach C ∈ C' do
11             if u ∈ C then
12                 degree_C[v] += 1;
13             if v ∈ C then
14                 degree_C[u] += 1;
15             if u ∈ C then
16                 C.put(v);
17             if v ∈ C then
18                 C.put(u);
19             processedElements += 1;
20             if processedElements mod W == 0 then
21                 C ← prune(C, s, degree_V, degree_C);
```

Algorithm 2: pruneComm

```
1  Function pruneComm (C, s, degree_V, degree_C)
2      minheap ← [];
3      foreach c ∈ C do
4          cp(c) = degree_C[c] / degree_V[c];
5          if minheap.size() < s then
6              minheap.push((c, cp(c)));
7          else if cp(c) > minheap[0] then
8              minheap.pop();
9              minheap.push(c, cp(c));
10     return set(minheap);
```

The second one is a stream $S = \langle e_1, e_2, \ldots, e_m \rangle$, where $e_i \in E$, and E is the set of edges of the undirected graph $G = \{V, E\}$ that S defines. 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. This output is available *on-demand* at all times as we process the stream.

Initialization: The first step of CoEuS is to initialize the communities using the seed-sets (Lines 2-6). This is a simple procedure in which we create 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 describe 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.

Stream processing: After initializing the communities, CoEuS is ready to process the stream (Lines 7-21). Due to the size of the network, we consider that maintaining the whole graph is prohibitive. Instead, we focus on the degree of each node in the graph as well as its degree in each community, and the nodes that comprise each community. For each incoming edge of the stream, we first increment by 1 the degree of each of the adjacent nodes in the graph (Lines 8-9). Then, for each community we wish to extend, we examine whether each of the adjacent nodes is a member of the community. If this is the case, we increment the community degree of the other node. In addition, if the other node is not a member of the community, we add the node to the community (Lines 11-18). 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. After that, CoEuS will examine for every community if nodes 9, or 8 are members of the community. This is true regarding node 8 for two communities. Therefore, CoEuS will increment the community degree of node 9 by 1 for both communities. In addition, CoEuS will add node 9 to both communities that node 8 belongs to.

As the diameters that real-world networks exhibit are small and in many cases decrease as the network grows [14], the communities CoEuS detects through the above process often grow considerably in size. However, 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 W. During a window, the communities may grow freely in size, as new edges arrive. However, when the window closes, CoEuS prunes all communities and keeps only the most *highly involved* nodes of each community (Lines 20-21). This process is detailed with Algorithm 2 and function *pruneComm*, which uses Eq. 2 to evaluate each node's participation level to community C. For each node $c \in C$ we calculate $cp(c)$ (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 the current node in it (Lines 8-9). The function outputs a set that comprises the nodes that remained in the min-heap after examining all the nodes of the community (Line 10). CoEuS prunes communities to a size of 100, as related studies state that quality communities do not surpass 100 nodes [26]. Moreover, CoEuS uses a window of 10,000 edges, a value derived via extensive exploratory testing that consistently works well.

Termination: CoEuS can be stopped at will, as the member nodes of each community are available at any moment. In the pseudocode of Algorithm 1, we consider a finite stream and CoEuS terminates when all elements of the stream have been

Algorithm 3: addToCommByEdgeQuality

```
1  Procedure addToCommByEdgeQuality
2      foreach C ∈ C' do
3          if u ∈ C then
4              degree_C[v] += degree_C[u] / degree_V[u];
5          if v ∈ C then
6              degree_C[u] += degree_C[v] / degree_V[v];
7          if u ∈ C then
8              C.put(v);
9          if v ∈ C then
10             C.put(u);
```

processed. However, CoEuS can handle infinite streams as well. Besides, all nodes of each community 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 the community.

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

For every edge of a graph stream, CoEuS examines whether an adjacent node is a member of a community. If so, CoEuS increments the respective community degree of its adjacent node by 1. This procedure takes under consideration the number of adjacent nodes each node has in a community to estimate the participation of the node in the latter. However, we do not consider the level of involvement of the adjacent nodes in the 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. We discuss here a variation of CoEuS to improve over a simple community degree measure by taking into account the edge quality of nodes w.r.t. each community. This variation is reminiscent of PageRank, that employs the network's link structure to improve over the in-degree measure [3].

Our variation employs Eq. (2), instead of incrementing the community degree of a node by 1 for all of its adjacent nodes that are members of a community. Eq. (2) is equal to the fraction of the adjacent nodes of a node that are also members of the community in question. This fraction is essentially 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. If this value is high, then the probability that an adjacent node is a member of the community is also high. Incrementing the community degree of a node using the value of Eq. (2) of its adjacent node instead of 1, enables CoEuS to *maintain its focus* in the community. In particular, this 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, the potential of nodes exhibiting low values of Eq. (2) to shift the focus of the community is limited.

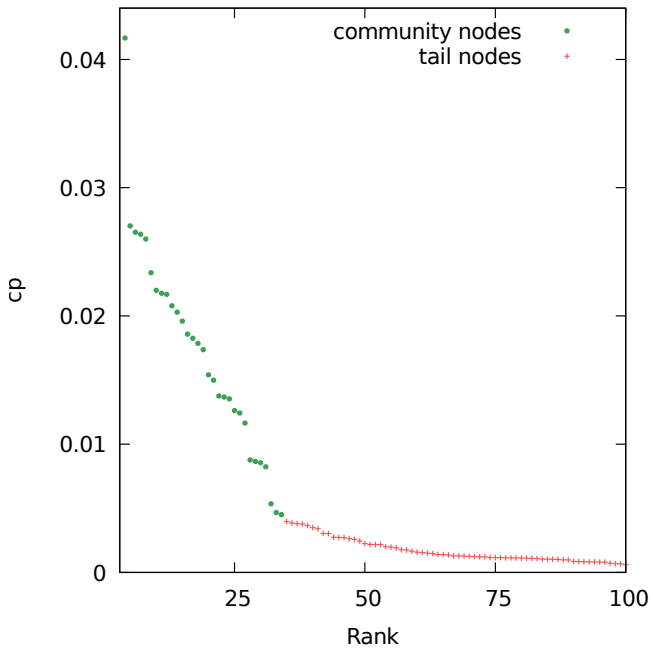

Fig. 3: Ranking of nodes according to their community participation values and the partitioning that Algorithm 4 makes to come up with a community automatically for a random citation network community.

Algorithm 3 details the above outlined approach, and can replace Lines 9-17 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 an estimation of Eq. (2) for the respective adjacent node.

E. 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 additionally solve the issue of determining the size of a community automatically and removing any irrelevant nodes.

Algorithm 4 details $dropTail$, 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 found 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 on both the graph and the community in question. Thus, setting a constant *threshold* value and discarding nodes that exhibit

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

Graphs	Type	Nodes	Edges	Average Degree
Amazon	Co-purchasing	334,863	925,872	2.76
DBLP	Co-authorship	317,080	1,049,866	3.31
Youtube	Social	1,134,890	2,987,624	2.63
LiveJournal	Social	3,997,962	34,681,189	8.67
Orkut	Social	3,072,441	117,185,083	38.14
Friendster	Social	65,608,366	1,806,067,135	27.53

Algorithm 4: dropTail

```
1  Procedure dropTail
2      Ĉ ← reverseSort(C);
3      totalDifference ← 0;
4      previous ← 0;
5      foreach c ∈ Ĉ do
6          if previous > 0 then
7              totalDifference ← cp(c) − previous;
8          previous ← cp(c);
9      averageDifference ← totalDifference / (Ĉ.size()−1);
10     previous ← 0;
11     foreach c ∈ Ĉ do
12         if previous > 0 then
13             difference ← cp(c) − previous;
14         previous ← cp(c);
15         if difference < averageDifference then
16             Ĉ.remove(c);
17         else
18             break;
```

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, $dropTail$ calculates the average distance between two consecutive nodes with regard to their ranking by their associated community participation values (Lines 5-9). Then, $dropTail$ iteratively examines 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, $dropTail$ stops, as it has spotted a significant gap between the values of two consecutive nodes (Lines 11-18). The result of this process for our example is also illustrated in Figure 3. 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 35th. Therefore, $dropTail$ considers that the tail of irrelevant nodes begins from the 35th 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, $dropTail$ does not consider seed nodes, but only those nodes that have been added to the community during its expansion process.

III. 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 proceed with the evaluation of CoEuS 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?
- Is CoEuS able to automatically determine the *size* of a detected community using our novel *dropTail* clustering procedure?
- Is the accuracy achieved through CoEuS *comparable* to that of state-of-the-art local community detection methods that use the entire graph?
- What are the merits of CoEuS with regard to *execution time* as well as *space efficiency* when compared to prior efforts?

A. Experimental Setting

Our dataset comprises the six publicly available social, co-authorship, and co-purchasing networks listed in Table I.[2] 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 a state-of-the-art algorithm [16] we have adopted its experimental setting and use the top-5000 ground-truth communities of each network that possess the highest quality according to [15], after enforcing a minimum community size of 20.

We implemented CoEuS using Java 8. Our implementation, as well as execution tests that enable the reproducibility of our results are publicly available.[3] 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 14.04.5 (Trusty Tahr) x86 64 OS. To maintain node and community degrees we employ in all our experiments Count-Min sketches. The latter are initialized with the following

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

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

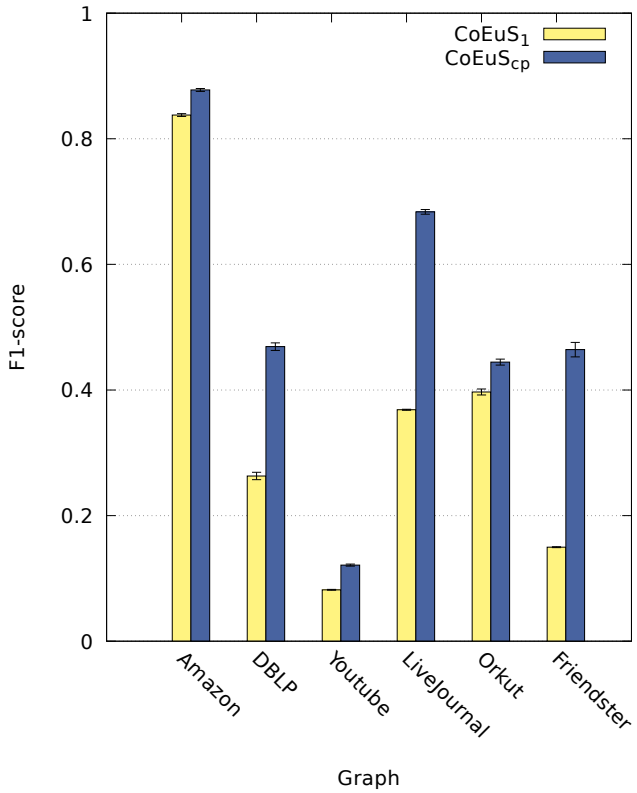

Fig. 4: F1-score comparison for CoEuS when incrementing community degree by 1 (CoEuS$_1$) and by community degree of the adjacent node (CoEuS$_{cp}$). The variation of CoEuS$_{cp}$ clearly improves the F1-score for all graphs our dataset. The improvement is particularly impressive for *orkut* and *dblp* graphs.

parameters: $i)$ $d = 7$, and $ii)$ $w = 200{,}000$, so that we obtain 99% confidence that $\epsilon < 0.00001$. Our evaluation assumes that three 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.

B. Impact of the Edge Quality Variation

We begin by studying the behavior of CoEuS when considering our two different techniques of incrementing the community degree of a node. We denote our algorithm as CoEuS$_1$ when the community degree of each node is incremented by 1 for every adjacent node found in the community (Algorithm 1) and CoEuS$_{cp}$ when the community degree of each node is incremented by the community degree of the adjacent node (Algorithm 1 with the edge quality variation of Algorithm 3).

Figure 4 illustrates a comparison between CoEuS$_1$ and CoEuS$_{cp}$ on their accuracy on detecting the ground-truth communities of the networks in Table I. It is clearly evident that the edge quality variation we introduce to CoEuS heavily impacts the ability of our algorithm to accurately retrieve the members of a community. We see that CoEuS$_{cp}$ achieves an increased F1-score compared to CoEuS$_1$ for all graphs included in our dataset. The improvement for graphs *dblp*, *livejournal*, and *friendster* is particularly impressive, increasing from 0.263 to 0.469, from 0.369 to 0.684, and from 0.15 to 0.464, respectively. Significant improvements with regard to F1-score are also achieved for graphs *orkut*, *amazon*, *youtube*, for which the F1-scores increase from 0.397 to 0.444, from 0.838 to 0.878, and from 0.082 to 0.121, respectively.

These results verify emphatically that the variation of Algorithm 3 successfully *favors* nodes that are actual members of the community in question, and *penalizes* nodes that exhibit weak ties 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 Algorithm 1.

We note that this experiment considers for both CoEuS$_1$ and CoEuS$_{cp}$ as size of each resulting community the size of the respective ground-truth community. We now proceed with 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.

C. 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 each community. The first one, i.e., Algorithm 2, is a pruning procedure that is periodically applied to reduce the size of the community. The second one, i.e., 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$_{cp}$ and CoEuS$_{cp\text{-}auto}$; for CoEuS$_{cp}$ we assume that the size of each community is known a priori, whereas CoEuS$_{cp\text{-}auto}$ automatically derives the size of a community using Algorithm 4.

The first two bars of Figure 5 illustrate the F1-scores achieved through CoEuS$_{cp}$ and CoEuS$_{cp\text{-}auto}$ when detecting the ground-truth communities of the networks of our dataset. As we can see, our $dropTail$ clustering algorithm is able to offer impressive performance, as the difference between the F1-score of CoEuS$_{cp}$ is in most cases negligible. More specifically, the difference in F1-score is under 0.06 for all networks of our dataset and 0.04 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 one employed in [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.

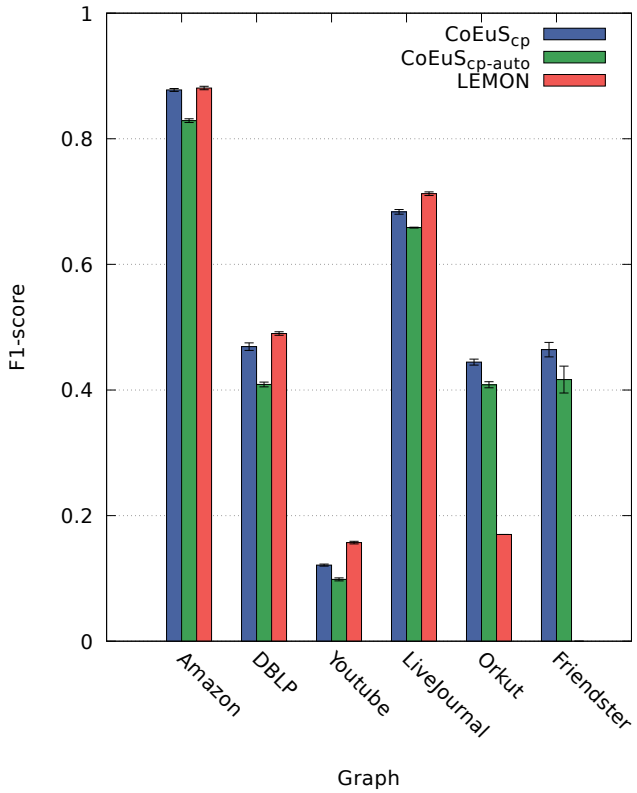

Fig. 5: F1-score comparison between LEMON and CoEuS.

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

After evaluating the impact of Algorithms 3 and 4 on the accuracy of CoEuS, and having verified their effectiveness in detecting communities in a variety of networks, 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 as it is shown in [16] to outperform all [12], [13], [23], whereas the more recent SCODA [11] reports significantly lower F1-scores and does not allow overlaps.

1) F1-score comparison: We begin the comparison of CoEuS$_{cp\text{-}auto}$ with LEMON as far as their accuracy on detecting communities is concerned. We initialize LEMON with three random seeds of each ground-truth community of our dataset and report here averages of multiple executions. The third bar of Figure 5 illustrates the F1-scores achieved with LEMON for all six networks. Note, that we were unable to retrieve results for the two largest networks of our datasets due to LEMON's memory requirements. However, we include here the results reported for *orkut* in [16].

As we can see, using the entire graph, LEMON is usually able to slightly outperform CoEuS$_{cp\text{-}auto}$ for small graphs. However, our algorithm is extremely competitive w.r.t. accuracy, despite the fact that it operates on a graph stream setting. More specifically, The average F1-score difference of CoEuS$_{cp\text{-}auto}$ with LEMON for the four smaller graphs of our dataset is 0.061. Regarding *orkut*, CoEuS$_{cp\text{-}auto}$ is far more accurate achieving an F1-score of 0.408 against LEMON's 0.17. Finally, our algorithm is able to achieve a notable F1-score of 0.417 for the largest graph of our dataset, which LEMON fails to handle due to its size.

TABLE II: Execution time comparison between CoEuS and LEMON for all graphs of our dataset. CoEuS is remarkably fast, even for the largest network of our dataset and clearly outperforms LEMON.

Graphs	CoEuS	LEMON
Amazon	0.0458 sec	3.1197 sec
DBLP	0.0575 sec	7.2756 sec
Youtube	0.176 sec	11.3834 sec
LiveJournal	1.573 sec	28.14 sec
Orkut	7.5171 sec	–
Friendster	158.6547 sec	–

These results are particularly impressive as the graph stream setting that CoEuS adheres to, is much more restrictive than the setting LEMON and other prior seed-set expansion methods operate on. CoEuS processes each edge of the graph as it is becoming available and maintains very limited information for each node and community. Hence, it is surprising that our algorithm achieves comparable accuracy to methods that utilize the entire graph structure. Furthermore, it is evident in Figure 5 that our effective novel graph stream techniques enable CoEuS to easily scale to large graphs, which other community detection methods fail to handle.

2) Execution time and space efficiency comparison: Having shown that CoEuS is competitive or better than state-of-the-art non-streaming algorithms as far as accuracy is concerned, we now report results concerning *execution time* and *space* efficiency.

Table II illustrates a comparison on the execution time between CoEuS$_{cp\text{-}auto}$ and LEMON. Regarding CoEuS$_{cp\text{-}auto}$, we consider a streaming setting in which we process every edge of each graph to update our structures and expand the communities in question. Similarly for LEMON, we read the entire graph in-memory, as only then we can sequentially expand each community in question. We use only one CPU core for both algorithms as parallel execution is not an option for LEMON which is written in Python. The results concern the average time needed for one community of each network. As we can see in Table II, our algorithm is extremely faster. In particular, CoEuS$_{cp\text{-}auto}$ is able to detect a community in the four smaller networks in time less than 2 seconds, whereas LEMON needs up to 28.14 seconds. Moreover, CoEuS is able to scale to networks reaching billions of edges requiring only an impressive total of 158.66 seconds for a community of the *friendster* network.

We note that 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. In particular, CoEuS is able to return the communities in question *on-demand* as we process the stream in *real-time*. The measurements reported in Table II additionally consider

TABLE III: Comparison of space requirements between CoEuS and LEMON for all graphs of our dataset. CoEuS uses two Count-Min sketches to hold a graph's elements and therefore its requirements depend only on the desired approximation quality of the sketches. LEMON maintains the adjacency lists of a graph and thus requires significantly more space.

Graphs	CoEuS	LEMON
Amazon	21.36MB	155.74MB
DBLP	21.36MB	156.49MB
Youtube	21.36MB	457.62MB
LiveJournal	21.36MB	2,652.99MB
Orkut	21.36MB	–
Friendster	21.36MB	–

edge processing, which we expect that in a real-life setting will happen 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 II indicate that CoEuS is a very attractive option even for non-streaming settings.

Table III shows a comparison between the two algorithms on their space requirements concerning the graph structure. $\text{CoEuS}_{\text{cp-auto}}$ employs two Count-Min sketches which are initialized in our experiments to require the same space regardless of the graph. Therefore, the space requirements are independent of the size of the graph and depend only on the desired approximation quality of the sketches and the number of communities in question. In contrast, LEMON needs to maintain in-memory the entire graph structure, using adjacency lists. Both algorithms require additional space to hold the communities we seek; however, this space is linear to the number of the communities and fairly insignificant compared to the graph structure.

It is evident that the space requirements of CoEuS are remarkably low. Even the largest graph of our dataset, reaching up to 1.8 billion edges is handled appropriately with just 21.36MB. In contrast, the space requirements of LEMON grow with the number of edges of a graph. The largest graph we are able to handle is *livejournal* with 34 million edges for which more than 2,500MB are needed. The two largest graphs of our dataset result in memory errors.

IV. Related Work

Our work lies in the intersection of community detection over streaming graphs and local community detection via seed-set expansion. We outline here pertinent aspects of these two areas.

Local community detection via seed-set expansion: Numerous recent approaches depart from the direction of working on 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 [12] propose a deterministic local algorithm to compute heat kernel diffusion and study the communities it produces when initiated with seed nodes. The authors compare with PageRank diffusion on large scale real-world graphs and report that using Heat Kernel diffusion they are able to detect smaller, more accurate communities, with slightly worse conductance. LEMON [16] 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 [16] employ the measure of conductance. In particular, they measure the conductance of the community as they increase its size, and stop at the first relative minimum conductance encountered. Both the above approaches are similar to our setting as they expand a seed-set of nodes into a community. However, neither of them is able to handle graph streams. LDLC [17] 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. [27] 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 [28] 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 recomputation after every update at the cost of slightly worse community quality. Our approach is different as we assume that we cannot maintain the whole graph in-memory, whereas the incremental adjustments that [28] 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. [11] 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 we pick 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 the more practical

assumption that the edges of the graph arrive at no particular order.

V. Conclusion

In this paper we propose and develop 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 to CoEuS, we propose two algorithms to improve the effectiveness of our approach. The first one 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 one 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 networks from various domains we show that CoEuS is able to offer accuracy that is equivalent to or better than that of methods exploiting the entire graph, even though it operates on a graph stream. The two algorithms we propose to enhance CoEuS, contribute enormously to its effectiveness and efficiency, 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. Therefore, 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.

References

[1] 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.

[2] 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.

[3] S. Brin and L. Page, "The Anatomy of a Large-Scale Hypertextual Web Search Engine," *Computer Networks*, vol. 30, no. 1-7, pp. 107–117, 1998.

[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] 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.

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

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

[8] 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.

[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 Int. Conf. on Knowledge Discovery and Data Mining*, 2012, pp. 597–605.

[10] K. He, Y. Sun, D. Bindel, J. E. Hopcroft, and Y. Li, "Detecting overlapping communities from local spectral subspaces," in *IEEE International Conference on Data Mining, Atlantic City, NJ, USA*, 2015, pp. 769–774.

[11] A. Hollocou, J. Maudet, T. Bonald, and M. Lelarge, "A linear streaming algorithm for community detection in very large networks," *ArXiv e-prints*, Mar. 2017.

[12] K. Kloster and D. F. Gleich, "Heat kernel based community detection," in *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA*, 2014, pp. 1386–1395.

[13] I. M. Kloumann and J. M. Kleinberg, "Community membership identification from small seed sets," in *Proc. of the 20th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, pp. 1366–1375.

[14] 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 Int. Conf. on Knowledge Discovery and Data Mining, Chicago, Illinois, USA, August 21-24, 2005*, pp. 177–187.

[15] 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 Int. Conf. on World Wide Web*, ser. WWW '08, 2008, pp. 695–704.

[16] Y. Li, K. He, D. Bindel, and J. E. Hopcroft, "Uncovering the small community structure in large networks: A local spectral approach," in *Proc. of the 24th Int. Conf. on World Wide Web*, 2015, pp. 658–668.

[17] P. Liakos, A. Ntoulas, and A. Delis, "Scalable link community detection: A local dispersion-aware approach," in *2016 IEEE Int. Conf. on Big Data, BigData 2016, Washington DC, USA, December 5-8, 2016*, pp. 716–725.

[18] P. Liakos, K. Papakonstantinopoulou, and A. Delis, "Memory-optimized distributed graph processing through novel compression techniques," in *Proc. of the 25th ACM Int. Conf. on Information and Knowledge Management, CIKM 2016, Indianapolis, IN, USA, October 24-28, 2016*, pp. 2317–2322.

[19] 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.

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

[21] 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.

[22] P. Pons and M. Latapy, "Computing communities in large networks using random walks," in *Computer and Information Sciences-ISCIS 2005*, 2005, pp. 284–293.

[23] J. J. Whang, D. F. Gleich, and I. S. Dhillon, "Overlapping community detection using seed set expansion," in *Proc. of the 22nd ACM Int. Conf. on Information & Knowledge Management*, 2013, pp. 2099–2108.

[24] J. Yang and J. Leskovec, "Community-affiliation graph model for overlapping network community detection," in *Proc. of the 12th IEEE International Conference on Data Mining*, 2012, pp. 1170–1175.

[25] ——, "Overlapping community detection at scale: a nonnegative matrix factorization approach," in *Proc. of the 6th ACM int. Conf. on Web Search and Data Mining*, 2013, pp. 587–596.

[26] ——, "Structure and overlaps of ground-truth communities in networks," *ACM Transactions on Intelligent Systems and Technology*, vol. 5, no. 2, p. 26, 2014.

[27] S. Yun, M. Lelarge, and A. Proutière, "Streaming, memory limited algorithms for community detection," in *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, 2014, pp. 3167–3175.

[28] A. Zakrzewska and D. A. Bader, "A dynamic algorithm for local community detection in graphs," in *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, ASONAM 2015, Paris, France, August 25 - 28, 2015*, 2015, pp. 559–564.