Graph Clustering and Community Detection

Matteo Pinna
matteo.pinna@hotmail.com
Technical University of Munich
Munich, Germany

ABSTRACT

Clustering and community detection are important research topics that have gained significant attention from researchers in recent years. The study of these topics has various applications in diverse fields such as computer science, biology, and social sciences. Thanks to research efforts in the past years, several algorithms to perform community detection have been developed, where the primary goal of these algorithms is to identify groups of nodes that are more densely connected to each other than to the rest of the network. For instance, most algorithms are based on the optimization of some measure related to the quality of the detected clusters, such as modularity or conductance, and consequently, aim at solving an optimization problem based on such metrics. However, most approaches fail to scale to large networks and are computationally intensive. This is a significant drawback as real-world graphs are massive in the number of vertices and edges. To address this scalability challenge, recent research has focused on developing highly scalable community detection algorithms by leveraging a streaming model applied to the graph analytics domain, allowing the processing of large-scale graphs in an efficient and scalable manner. In this paper, an overview of the research topic of community detection in graphs is provided, as well as the state-of-the-art of community detection algorithms. Then, two relatively recent and stream-based algorithms of choice are going to be highlighted and compared.

CCS CONCEPTS

• **Computer systems organization → Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability.

KEYWORDS

networks, graph clustering, community detection, streaming model

ACM Reference Format:
Matteo Pinna. . Graph Clustering and Community Detection. In *Proceedings of (Advanced Seminar: Large-scale Graph Processing and Partitiniong).* ACM, New York, NY, USA, 8 pages.

1 INTRODUCTION

Graphs are a powerful tool for modeling, analyzing, and understanding collections of relationships between entities, such as social ties between people or links between web pages. Their relevance is demonstrated by their widespread applications in several domains, from computer science [17] to physics [19] and biology [10]. In data analysis, it is often necessary to focus not only on individual entities within a dataset but also on the connections between them (i.e., *network problem*). In such cases, the use of graphs as a mathematical representation of the entities as vertices and connections between them as edges is an effective approach.

The primary purpose of using a graph to represent a network can vary depending on the specific research field or real-world scenario being considered. For instance, they may be used to perform sentiment analysis on social media [2] or to serve as the main storage format in a graph database [4] [16]. In the graph analytics domain, the development and study of efficient algorithms to identify clusters of vertices that are connected, according to some heuristic, measure, or definition of similarity, and separate them from those which do not correlate have become quite an active and challenging research area. This task is typically referred to as *community detection* [8] or *graph clustering* [18] [14], where the identified groups of vertices would be labeled as clusters or communities, respectively.

Real-world networks are often both massive, with millions of vertices and billions of edges, and dynamic, with the potential for the graph structure to evolve over time due to edge insertion and/or deletion. Some examples are the *web graph*, with vertices being web pages and edges being hyperlinks between them, *social media graphs*, with vertices being users and edges being connections of various forms between them, or the *Bitcoin transaction graph*, with vertices being Bitcoin addresses and edges being transactions between them. In such cases, traditional clustering algorithms can be inefficient and struggle in terms of scalability due to the massive size of the networks being considered. To address these scalability issues, the *data stream model* [6] [15], which allows data to be processed as a stream of edges - without storing the entire dataset in memory - has become quite popular in the last decade. This paper provides an overview of graph clustering and community detection as research topics and the domains in which they are often applied. The focus is then shifted to state-of-the-art techniques used in the design of clustering algorithms. Finally, two efficient and relatively recent approaches for community detection based on the streaming model are going to be presented and compared.

1.1 Paper outline

The following sections are organized as follows. First, the problem as a whole of graph clustering and community detection on graphs will be discussed, to give an overview of the motivations, application domains, and related challenges. Then, the state-of-the-art of (streaming) clustering algorithms is going to be reviewed. Finally, two algorithms are going to be discussed, SCoDA, a linear streaming algorithm for community detection introduced by Hollocou et al. [2017], based on a simple probabilistic idea and which can handle static graphs in an offline fashion, and CoEuS, introduced by Liakos et al. [2017], which can handle static graphs in an online fashion and is based on seed-set expansion and a novel introduced quality measure to detected relevant nodes within a community.

2 STATE OF THE ART

Graph clustering and community detection are often used interchangeably in the literature, as both terms refer to techniques that aim to identify groups of vertices that share common properties and/or play similar roles within the graph [8]. At the same time, it is possible to find literature where a subtle difference between the two is identified [11]. In general, we could state that graph clustering refers to a broad concept of partitioning vertices of a graph based on some measure of similarity, where the definition of similarity may vary depending on the context or approach being used (e.g., vertices' attributes, vertices' behavior, etc.). Community detection, on the other hand, could be defined as a specific type of graph clustering that focuses on identifying groups of vertices that are more densely connected between each other w.r.t. the rest of the graph, by taking into account the graph's structure and the edges' distribution. In the following sections, for simplicity, the two terms will be used interchangeably to refer to the general problem of identifying groups of vertices within a graph.

The interest in community detection as a research topic has seen a huge increase in the past years. This increase in popularity can be attributed, as mentioned in [5] [1], to the growth in graph size and the increasing forms of applications for graph data. Several community detection algorithms, based on a variety of different approaches and core principles, exist for identifying communities in graphs. Prominent techniques include quality measure optimization, spectral clustering, and random walk-based approaches. [8]. One thing that should be noted is that no universally accepted definition of what a (good) community is can be found in the literature, as there are several different approaches that rely on different principles and metrics to identify (good) groups of vertices. In general, most algorithms rely on the principle that a good community should be very densely connected internally (i.e., have a high number of internal edges) and loosely connected or at least less connected w.r.t. the rest of the graph (i.e., have a lower number of external edges).

Among the class of algorithms that rely on quality metrics, *modularity* optimization is probably the most popular. Modularity, introduced by Newman and Girvan, assesses the relative density of edges within a sub-graph compared to the density that would be expected in a randomly generated graph with the same degree distribution. Since high values of modularity supposedly indicate a "good" community, these algorithms seek to maximize the modularity score. Another quite relevant quality measure is the *conductance*, defined as the ratio between the number of edges connecting a community to the rest of the graph (i.e., cut size) and the number of edges inside the community (i.e., volume). In the case of conductance, it is trivial to notice that a lower value should identify a better community since the nominator takes into account the number of internal edges which is expected to be high in a cohesive community. Random walks are a probabilistic technique where the graph is traversed by a random walker which randomly selects, at each step, a vertex and moves to it until a stopping criterion is reached. Random walk-based algorithms, on the other hand, focus on the idea that a random walker is likely to spend a prolonged period of time inside a community due to the high number of internal edges that is expected if the community is of good quality. Finally, spectral clustering identifies clusters of vertices in a graph based on the eigenvectors of the graph's Laplacian matrix. The Laplacian matrix of a graph encodes information about the graph's structure and connectivity. It is computed by subtracting the matrix with nodes' degrees on the diagonal, representing the graph structure, and the adjacency matrix where each cell is either 0 or 1 if the nodes ids represented by the corresponding row and column are connected by an edge or not, respectively, hence representing the graph's connectivity. The intuition is that the eigenvectors will have large values if the nodes are in the same community, and low values in the other case, therefore it is possible to group vertices with similar eigenvector values. Spectral clustering is particularly popular since it has been demonstrated how the technique is robust w.r.t. noise and also capable of identifying non-convex clusters. There are several other techniques that may be less popular, depending on the specific domain of application, which could be worth mentioning, such as seed-set expansion [3] [20], which consists of initializing one or more seed-sets with a (typically) small number of nodes that supposedly belong to the same community, then the communities are expanded by processing additional information from the graph and adding nodes that are similar to the ones inserted during the seed-set expansion phase. Also clique percolation by Palla et al., which is a technique specifically designed to detect overlapping communities in graphs, based on the idea that the internal edges of a community are likely to form cliques due to their high density, while also being true that external edges are not likely to form a clique.

Independently of the approach being used, being able to scale up to massive graphs while also providing satisfactory performance is not an easy task. This is a crucial issue since, as previously mentioned, most real-world networks, but also artificial ones, are massive. In order to address this scalability issue when handling very large networks the *data stream model* [6] has gained a lot of attention. The data stream model applied to graph analytics allows the graph to be processed as a stream of edges, avoiding the storage of the entire graph in memory. Depending on the streaming model being used, it is possible to differentiate between stream-based community detection for static (insert-only) streams, where the stream contains only edge insertions, and dynamic streams, where the stream consists of both edge insertions and deletions. It is trivial to notice how dynamic graphs, and therefore dynamic streams of edges, pose an additional challenge in the design of these algorithms since the structure of the graph changes over time and previously identified communities may be removed due to an edge deletion. As such, it is important to consider the difference in data stream models, as algorithms designed for static graphs may not be suitable for dynamic graphs and vice-versa.

3 SCODA

In this section, SCoDA [9], a linear time, linear space, (insert-only) stream-based, algorithm for community detection on large-scale networks, is discussed. By leveraging the stream-model SCoDA is able to address the scalability issues that arise when considering massive graphs.

The main idea upon which the algorithm is based is that if an edge e is randomly chosen then it is more likely for the adjacent vertices to belong to the same community, (i.e., e is an *intra-community* edge), and less likely that they belong to different communities (i.e., e is an *inter-community* edge). Therefore, if a random permutation of the stream is considered, it is expected for intra-community edges to arrive *early* w.r.t. inter-community edges, which are expected to arrive *late*. An edge will be considered to be early if the degrees of the adjacent vertices are low while taking into account only the previously arrived edges to compute such node degrees. Since this randomness in the stream processing is needed for the assumption to hold, the stream is shuffled and a corresponding random permutation is generated during a pre-processing phase before the stream itself is iterated over.

3.1 Notation

Before introducing the algorithm pseudo-code, some notation is required. Consider an un-directed and un-weighted graph $G(V, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$ and $E = \{e_1, e_2, \ldots, e_m\}$ are the set of vertices/vertices and the set of edges, respectively. An edge will be represented as $e = (u, v)$ where $u, v \in E$. Then, the degree $d(u)$ of a vertex $u \in V$ is defined as the number of edges that are incident to u

$$d(u) = |\{v \in V \ : \ (u, v) \in E\}| \tag{1}$$

Note, in this case, there is no distinction between *indegree* (i.e., number of incoming edges) and *outdegree* (i.e., number of outgoing edges) since an un-directed graph is being considered.

3.2 Algorithm

The algorithm takes, apart from the stream of edges S, an input parameter $D \geq 1$, which will be used as a threshold to decide whether an edge arrived early or late, and an optional parameter p. The output to be returned is a set of communities $c = \{c_1, c_2, \ldots, c_l\}$, where l is the number of identified communities. Two array data structures d and c are used to store the degree and the associated community id for each vertex, respectively, as maintaining the entire graph would be prohibitive depending on the size of the network being considered.

In Algorithm 1 the pseudo-code of SCoDA. In the *initialization* (pre-processing) phase the two previously mentioned data structures are initialized, the degree of each vertex is set to 0 and each vertex is put in its own community, i.e. $\forall\, i \ : \ d_i = 0, c_i = i$, then the stream S is shuffled in order to generate a random permutation of S. In the *stream processing* phase, for a new edge $e = (u, v)$, the degrees of the adjacent vertices u, v are incremented and their communities updated according to the following cases:

- If the degrees $d(u), d(v)$ are above the threshold D (i.e., the edge is late) then no community update is performed;
- If the degrees $d(u), d(v)$ are both below the threshold D (i.e., the edge is early) then either the vertex with the lower degree joins the community of the other one or, in case of equality, an arbitrary choice can be made with probability p.

Algorithm 1: SCoDA

```
Input: (List of edges $E$ between vertices $\{1, \ldots, m\}$,
        parameter $D \geq 1$, and the probability $p$)
Output: ($c = \{c_1, \ldots, c_n\}$)
1  begin
       // Initialization
2      for $i = 1, \ldots, n$ do
3          $d_i \leftarrow 0$ and $c_i \leftarrow i$;
4      Shuffle the list of edges $E$;
       // Processing
5      for $j = 1, \ldots, |E|$ do
6          $(u, v) \leftarrow$ j-th edge of $E$;
7          $d_u \leftarrow d_u + 1$ and $d_v \leftarrow d_v + 1$;
8          if $d_u \leq D$ and $d_v \leq D$ then
9              if $d_u < d_v$ then
10                 $c_u \leftarrow c_v$;
11             else if $d_v < d_u$ then
12                 $c_v \leftarrow c_u$
13             else
14                 $choice \leftarrow rand(0, 1)$ if $choice \geq p$ then
15                     $c_u \leftarrow c_v$;
16                 else
17                     $c_v \leftarrow c_u$;
```

An important observation made by Hollocou et al. [2017] is the fact that their algorithm design does not propagate community updates to neighbors vertices when an edge is processed, i.e. if a new edge $e = (u, v)$ arrives only the community of e and v will (possibly) be updated. This characteristic makes SCoDA extremely parallel as its execution can be split into different nodes by assigning subsets of the set of vertices E to each node. Since the processing of each edge can only impact the degree of the edge being connected there won't be consistency problems across the nodes that will be operating on distinct subsets of edges.

3.3 Threshold parameter

The choice of D is crucial in the design of SCoDA, as it is the (only) parameter that determines whether a new edge causes a change in the pre-existing communities or not. Hence, the tuning of D heavily impacts the performance and results of the algorithm. The goal should be to avoid too low or too high values since a very high D would possibly cause a degradation in the quality of communities, as inter-community edges may cause community updates due to the not very selective threshold, while a very low D would possibly cause good communities to split into sub-communities, as it would be very often the case for an intra-community edge to be ignored due to the very selective threshold. Hollocou et al. [2017] propose three different options for tuning D:

- **Average degree:** the average value of the degree over the network

$$D = d_{avg}$$

- **Median degree:** the median value of the degree over the network
$$D = d_{median}$$
- **Mode of the degree distribution:** the *mode* of the degree over the network, i.e. the most common degree excluding leaf vertices
$$D = d_{mode}$$

In the original paper, it is shown through different benchmarks how the mode of the degree distribution typically provides better results w.r.t. the average degree and the median degree. Note that, the mode of the degree distribution d_{mode} can be defined as

$$d_{mode} = \underset{d>1}{\operatorname{argmax}} \ |\{u \in V \ : \ d(u) = d\}| \tag{2}$$

3.4 Complexity

Consider n and m to be the number of vertices and the number of edges, respectively. In terms of time complexity, the shuffling performed on the edge list is linear in m if an efficient algorithm is used (e.g. *Fisher-Yates* algorithm) and the initialization for d and c is linear in n. Finally, the main loop to iterate over the edges is also linear in m. Therefore, the algorithm time complexity is linear in m since it is often the case for the number of edges to be very large w.r.t. to the number of vertices (i.e., $m >> n$). In terms of space complexity, SCoDA only uses the two arrays d and c of size n. Hence, the space complexity is linear in n.

3.5 Considerations

SCoDA is a linear, compact, and elegant algorithm capable of processing large-scale static graphs in a streaming fashion. However, it is not designed to handle dynamic graphs that evolve over time as only insert-only streams have been considered in the original paper. This is a non-negligible limitation as real-world networks are typically dynamic. Another limitation of the algorithm is that in order to be able to generate the random permutation of the stream, it would be strictly necessary to know the entire graph structure in advance. Therefore, SCoDa cannot be applied in a scenario where the graph is not (completely) known a priori, i.e. it cannot be applied in an online fashion.

4 COEUS

In this section, CoEuS [13], a streaming, sub-linear space, seed-set expansion based, algorithm for community detection on large-scale networks, is discussed.

4.1 Notation

Liakos et al. [2017] introduce a novel measure to identify good communities, the *community participation* score. The community participation of a vertex $u \in V$ in a community C represents the participation level of that vertex in a specific community and is defined as the ratio between the number of edges going from a vertex in the graph to a vertex in the community C, and the total number of edges:

$$cp(u) = \frac{|\{(u, v) \in E \ : \ v \in C\}|}{|\{(u, v) \in E\}|}. \tag{3}$$

Intuitively, since community participation is a representation of the participation of a vertex in a community, it is expected that including vertices with high cp values in a community C will result in a low value of conductance, hence a good community.

4.2 Algorithm

The algorithm requires, apart from the stream of edges S, a single input parameter K', i.e. the set of community seed-sets that will be used to initialize the corresponding communities. Consider $K' = \{K_1, K_2, \ldots, K_s\}$ with $K_i = \{k_1, k_2, \ldots, k_l\}$, where s and l are the number of seed-sets and the number of nodes in each seed-set, respectively. The output is a set containing the identified communities. Three data structures are used to store the degree of each vertex in the graph, the community degree of each vertex in each community, and the set of vertices for each identified community.

In Algorithm 2 the pseudo-code of CoEuS is presented (Algorithm 1 in [13]), exception for some comprehensibility adjustments. Note, $degree_V$ keeps track of the degrees for all vertices in the graph, $degree_C$ keeps track of the community degrees for all vertices in the graph, and for each community C, finally C' stores the identified communities. In the *initialization* phase, communities are initialized using the provided seed-sets, which should have been carefully selected beforehand using appropriate techniques as they heavily influence the algorithm's output. Each community C_i is initialized with the vertices contained in the corresponding seed-set $K_i \in K'$. In the *processing* phase, the pre-existing (seed-set initialized) communities are incrementally expanded as the graph stream is processed. The degrees of the adjacent vertices in a newly arrived edge are incremented, besides, for each pre-existing community C, if a vertex belongs to that community also the corresponding community degree is incremented and the vertex itself is added to C. Finally, the *pruning* phase is where the algorithm focuses on removing irrelevant edges in each community and maintaining only those that are highly relevant, i.e. those that have high community participation values and are therefore highly involved in that specific community. For this purpose, a window of size W is used to decide when the pruning is to be applied, the window *processedElements* grows freely while the stream is being processed, and when it closes (i.e., reaches its full size when $processedElements \ \% \ W == 0$) the pruning is performed on each currently detected community.

In Algorithm 3 the pseudo-code related to the pruning technique (Algorithm 2 in [13]). It takes four parameters C, s, $degree_V$ and $degree_C$: the community to which the pruning is to be applied, the maximum size the community should have after the pruning, the set of degrees for each vertex, and the set of degrees for each vertex of each community, respectively. The output will be the remaining set of vertices to be stored in the community C as a result of the pruning. Note, the previously introduced measure of community participation cp is used here in order to determine when a vertex is highly involved in a community or not. The *pruneComm* function initializes a min-heap that will be used to store the vertices that should not be pruned. Then, for each vertex, $c \in C$ the community participation $cp(c)$ is computed, if the min-heap is not full, $minheap.size() < s$, the pair $(c, cp(c))$ is added to the data structure, otherwise we check whether $cp(c)$ is smaller than the minimum value in the min-heap, $cp(c) < minheap[0]$, as the min-heap is

Algorithm 2: CoEuS(S, K')

```
Input: (A graph stream $S$, and a set of community seed-sets $K'$)
Output: (A set of communities $C'$)
begin
    // Initialization (seed-sets)
    foreach $K \in K'$ do
        $C \leftarrow \{\}$;
        foreach $k \in K$ do
            $C[k] = 1$;
        $C'.put(C)$;
    // Processing
    while $\exists(u, v) \in S$ do
        $degree_V[u] += 1$;
        $degree_V[v] += 1$;
        foreach $C \in C'$ do
            if $u \in C$ then
                $degree_C[v] += 1$;
                $C.put(v)$;
            if $v \in C$ then
                $degree_C[u] += 1$;
                $C.put(u)$;
        $processedElements += 1$;
        // Pruning
        if $(processedElement \% W) == 0$ then
            foreach $C \in C'$ do
                $C \leftarrow pruneComm(C, s, degree_V, degree_C)$;
```

a data structure where elements are sorted in ascending order. If the latter case is true the vertex c is added to the min-heap with its respective community participation value in substitution of the previously smallest value $minheap[0]$ which instead is dropped.

Algorithm 3: pruneComm

```
Function pruneComm($C, s, degree_V, degree_C$):
    $minheap \leftarrow []$;
    foreach $c \in C$ do
        $cp(c) = \frac{degree_C[c]}{degree_V[c]}$;
        if $minheap.size() < s$ then
            $minheap.push(c, cp(c))$;
        else if $cp(c) > minheap[0]$ then
            $minheap.pop()$;
            $minheap.push(c, cp(c))$;
    return $set(minheap)$;
```

We're now going to present two possible optimizations, introduced by Liakos et al. [2017], that can be applied to CoEuS in order to improve the quality of the detected communities. The first is based on the idea of taking into consideration edge quality when updating community degrees, the second is based on the idea of reducing the identified community size by removing insignificant members and maintaining only those that are truly valuable to that community.

4.3 Edge quality optimization

In the original version of Algorithm 2, when an edge is processed the community degree of a vertex is incremented by 1 for each adjacent vertex that is a member of the community, hence to estimate the level of involvement of a vertex in a community only the number of its adjacent vertices in that community are considered. However, this approach does not take into account whether the adjacent vertices have a high level or low level of involvement in the community, i.e. a high community participation value or a low community participation value, respectively. The proposed optimization involves using the definition of community participation in Equation 3 in the update of the community degrees. Recall, the community participation $cp(v)$ of a vertex v was defined as the ratio between the number of edges connecting v to some vertex $u \in C$ where C is a community and the total number of edges in the graph. Hence it can be rewritten as the ratio between the community degree of u w.r.t. community C and the degree of u,

$$cp(u) = \frac{degree_C[u]}{degree_V[u]}.$$

This value can be interpreted as an estimate of the probability that a one-step random walk starting from the vertex will lead to a vertex that is a member of the community in question, hence why the community participation value of a vertex grows with its involvement in the community. A high value means that the probability that an adjacent vertex is a member of the community is also high. By applying such change the algorithm will favor vertices that are adjacent to well-established members of the community since their community degree will be significantly incremented. In contrast, vertices with low values of this measure will be provided with a smaller increment.

It is now possible to rewrite Algorithm 2 as Algorithm 4 where the edge quality optimization step is implemented by Procedure 5 (Algorithm 3 in [13]) and can replace lines 10-18 of the original implementation.

4.4 Community size optimization

The pruning optimization departs from the previous approach of using a fixed threshold for pruning communities based solely on their size, as the size of each community may vary. The idea is to sort the vertices of a community in descending order based on their community participation scores, and then drop those with weak ties to the community as their community participation score should be irrelevant w.r.t. to other vertices in the same community. However, since the distribution of community participation scores may vary depending on the specific community being considered, this approach has an adaptive nature and determines the actual size of a community and the score threshold after which applying the pruning in a fully automatic fashion.

This optimization is implemented in Procedure 3. A community C is first sorted in descending order based on the community participation score of each vertex, $\hat{C} = reverseSort(C)$. Then, in

Algorithm 4: CoEuSedgeQuality(S, K')

```
Input: (A graph stream $S$, and a set of community seed-sets $K'$)
Output: (A set of communities $C'$)
begin
    // Seed-sets initialization
    foreach $K \in K'$ do
        $C \leftarrow \{\}$;
        foreach $k \in K$ do
            $C[k] = 1$;
        $C^{'}.put(C)$;
    // Stream processing
    while $\exists(u, v) \in S$ do
        $degree_V[u] += 1$;
        $degree_V[v] += 1$;
        // Edge quality optimization
        $addToCommByEdgeQuality()$;
        $processedElements += 1$;
        // Pruning
        if $(processedElement \% W) == 0$ then
            foreach $C \in C'$ do
                $C \leftarrow pruneComm(C, s, degree_V, degree_C)$;
```

Algorithm 5: addToCommByEdgeQuality

```
Procedure addToCommByEdgeQuality():
    foreach $u \in C$ do
        if $u \in C$ then
            $degree_C[v] += \frac{degree_C[u]}{degree_V[u]}$;
            $C.put(v)$;
        if $v \in C$ then
            $degree_C[u] += \frac{degree_C[v]}{degree_V[v]}$;
            $C.put(u)$;
```

the *Computing average* phase the average difference between the scores of two consecutive nodes is computed. Finally, in the *Drop tail* step, the community is iterated over again but starting from the last node. If the difference is below the average, the vertex will be dropped from the community, the execution stops when a significant gap between the current difference and the average is found. Note, *dropTail* takes into consideration only nodes that have been added during the stream processing phase, this means that seed nodes are not being considered during this pruning.

4.5 Complexity

Consider n, m, and l to be the number of nodes, the number of edges, and the number of seed-set initialized communities respectively. The time complexity of CoEuS, as presented in Algorithm 2, is dominated by the loop for processing the stream and the loop that iterates over each community. Thus, the time complexity can be expressed as $O(m * l + k * l)$, where k represents the number of

Algorithm 6: dropTail

```
Procedure dropTail():
    $\hat{C} \leftarrow reverseSort(C)$;
    $totalDifference \leftarrow 0$;
    $previous \leftarrow 0$;
    // Computing average
    foreach $c \in \hat{C}$ do
        if $previous > 0$ then
            $totalDifference \leftarrow cp(c) - previous$;
        $previous \leftarrow cp(c)$;
    $averageDifference \leftarrow \frac{totalDifference}{\hat{C}.size()-1}$;
    $previous \leftarrow 0$;
    // Drop tail
    foreach $c \in \hat{C}$ do
        if $previous > 0$ then
            $difference \leftarrow cp(c) - previous$;
        $previous \leftarrow cp(c)$;
        if $difference < averageDifference$ then
            $\hat{C}.remove(c)$;
        else
            break;
```

executions of the *pruneComm* function and depends on the choice of W. The optimizations introduced in Algorithm 5 and Algorithm 6 do not affect the overall time complexity. In terms of space complexity, it is possible to store all the required information using l sets, one for each community, and $|V|(l+1)$ integers, where $|V| \cdot l$ integers are needed to store the degree of each vertex in each community's subgraph and $|V|$ integers are required to store the degree of each vertex in the graph. Note that, as the number of communities can be large, Liakos et al. [2017] propose using *COUNT-MIN sketches* [7], a well-known sub-linear space data structure that can handle high-rate updates and is ideal for summarizing data streams, implementation to efficiently and accurately summarize the required integers.

4.6 Considerations

CoEuS is a community detection algorithm that includes various optimization methods to enhance its accuracy as compared to its traditional implementation. However, its space complexity can be significant as the number of communities may be substantial in some instances, leading to a very large value for $|V| \cdot l$ which represents the number of integers needed to store the community degree for each vertex in each community. Nonetheless, the use of COUNT-MIN sketches proposed by the authors, allows CoEuS to achieve sub-linear space consumption. CoEuS is designed to manage static graphs and therefore, it does not account for edge deletions during stream processing. Additionally, the seed-set expansion is likely the most critical component of the algorithm as it heavily impacts the resulting detected communities. It is worth noting that seed-set initialization is a challenging task as it may be hard to determine the most suitable approach for initializing the seed-sets. Furthermore, there may be scenarios where executing seed-set initialization is not feasible since it requires a certain level of knowledge regarding the graph. More specifically, it is crucial to have knowledge of vertices for each community that are deemed good members, as this information is needed to include them in the seed-set of that particular community.

5 BENCHMARK AND COMPARISON

To evaluate the performance of the previously introduce algorithm and be able to present a comparison, the (average) *F1-score* [21] is considered. The F1-score measures the accuracy of a test by combining the results of *precision* and *recall*. The precision p and recall r can be defined as

$$p = \frac{TP}{TP + FP} \qquad r = \frac{TP}{TP + FN}$$

where TP and FP are the true positive and false positive respectively, and FN are the false negatives.

The F1-score can be defined as

$$F_1 = 2\frac{p \cdot r}{p + r}$$

We now adapt such definitions to define the precision, recall, and F1-measures w.r.t. a test on a true community C given a predicted community $\hat{C}$.

$$p(\hat{C}, C) = \frac{|\hat{C} \cap C|}{|\hat{C}|} \qquad r(C, \hat{C}) = \frac{|\hat{C} \cap C|}{|C|}$$

$$F_1(\hat{C}, C) = 2\frac{p(\hat{C}, C) \cdot r(\hat{C}, C)}{p(\hat{C}, C) + r(\hat{C}, C)}$$

Then, assuming the the graph contains k ground-truth communities, $C = \{C_1, \ldots, C_K\}$, and the communities detected by the algorithm are l, $\hat{C} = \{\hat{C}_1, \ldots, \hat{C}_L$, the F1-score between these two sets can be expressed as

$$F_1(\hat{C}, C) = \frac{1}{K}\sum_{k=1}^{K} \max_{1 \le l \le L} F_1(\hat{C}_l, C_k)$$

Finally, given 5 the average F1-score between set C and set $\hat{C}$ is

$$\bar{F_1}(\hat{C}, C) = \frac{F_1(\hat{C}, C) + F_1(C, \hat{C}}{2}$$

The performance of SCoDA and CoEuS is evaluated on various networks with ground-truth communities from the *SNAP (Stanford Large Network Dataset Collection)* dataset [12]. In Table 1 the datasets with their corresponding number of nodes, edges, and communities.

	# nodes	# edges	# communities
Amazon	334,863	925,872	311,782
DBLP	317,080	1,049,866	1,449,666
YouTube	1,134,890	2,987,624	8,455,254
LiveJournal	3,997,962	34,681,189	137,177
Orkut	3,072,441	117,185,083	49,732
Friendster	65,608,366	1,806,067,135	2,547

Table 1: SNAP dataset networks with ground-truth communities.

In Table 2 and Figure 1 the average F1-score comparison between SCoDA and the different implementations of CoEuS, where $CoEuS_{eq}$ and $CoEuS_{eq-dt}$ represent CoEuS with edge quality optimization and CoEuS with edge quality optimization and drop tail optimization. Note, while a comparison of the execution time and space is not included, as the results in the respective papers are dependent on the machine on which the algorithms were tested, a comprehensive theoretical analysis of their respective time and space complexity has been provided in the previous sections.

	SCoDA	*CoEuS*	$CoEuS_{eq}$	$CoEuS_{eq-dt}$
Amazon	0.37	0.83	0.87	0.83
DBLP	0.23	0.26	0.46	0.40
YouTube	0.24	0.082	0.12	0.098
LiveJournal	0.27	0.36	0.68	0.65
Orkut	0.42	0.39	0.44	0.40
Friendster	0.20	0.15	0.46	0.41

Table 2: Average F1-score comparison between SCoDA and CoEuS.

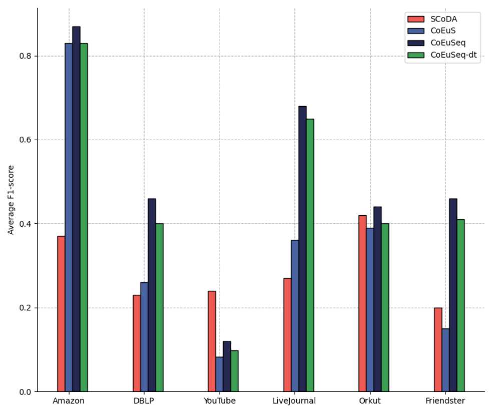

Figure 1: Average F1-score comparison between SCoDA and CoEuS.

CoEuS delivers noteworthy improvements in performance w.r.t. SCoDA, particularly when the edge quality optimization version $CoEuS_{eq}$ is considered, on the *Amazon*, *DBLP*, *LiveJournal*, and

Friendster data sets, where the average F1-score improves by 0.50, 0.23, 0.41 and 0.26, respectively. Surprisingly, CoEuS falls short when applied to the *YouTube* network and performs far worse than SCoDA, achieving an average F1-score even below a 0.1 threshold for both *CoEuS* and $CoEuS_{eq-dt}$ versions. The *Orkut* dataset is the only one where it is possible to observe quite similar results instead. Finally, it should be noted how the drop tail optimization performs weaker compared to when only the edge quality optimization is applied, despite the minimal difference in the results.

As both algorithms have been tested on real-world datasets, such as social networks (e.g. Friendster, Youtube) or the co-purchasing network of Amazon, it could be interesting to run some benchmarks on artificial networks to gain insights into their performance in a different setting.

6 CONCLUSIONS AND FUTURE WORK

The paper has presented the importance of community detection as an active and crucial research topic that aids in identifying groups of vertices sharing similar characteristics or roles in a graph. An overview of the state-of-the-art in community detection has been provided, focusing on different approaches and techniques used to identify communities in a graph, such as modularity optimization, spectral clustering, and random walk-based approaches. We have also discussed the issue of scalability when handling very large networks and addressed it by introducing the streaming model applied to graph processing.

In addition, two relatively recent stream-based community detection algorithms have been discussed and compared - SCoDA and CoEuS. This paper showed how SCoDA, a linear time and linear space algorithm, leverages the expected arrival times of intra-community and inter-community edges in a randomly shuffled stream of edges and how this core aspect is also one of its limitations, as it requires the entire graph to be available at once in order to be able to generate a random permutation of the edges. This makes SCoDa not suitable to work in an online fashion. In contrast, CoEuS, a sub-linear space, seed-set expansion-based algorithm, has no such limitations related to the order of edge arrival, making it more suitable for online processing. However, initializing the seed-sets can be a very challenging task and may not be feasible in cases where it is not possible to identify community members to put into each seed-set beforehand. Finally, the paper has presented a comparison between SCoDA and CoEuS using the average F1-score, a popular metric combining precision and recall, on ground-truth networks from SNAP datasets. The results show how CoEuS, when considering its optimizations, typically outperforms SCoDA in most scenarios.

In conclusion, community detection in graphs is a challenging topic, and recent research efforts have focused on developing efficient and scalable algorithms. Although significant progress has been made in recent years, there is still much to be accomplished, such as the development of novel approaches that can handle massive networks or the development of techniques for community detection in dynamic networks. Therefore, the development of novel approaches that can handle these challenges is crucial for advancing the field and enabling the analysis of complex networks.

REFERENCES

[1] Tariq Abughofa, Ahmed A. Harby, Haruna Isah, and Farhana Zulkernine. 2021. Incremental Community Detection in Distributed Dynamic Graph. In *2021 IEEE Seventh International Conference on Big Data Computing Service and Applications (BigDataService)*. 50–59. https://doi.org/10.1109/BigDataService52369.2021.00012

[2] Fotis Aisopos, George Papadakis, and Theodora Varvarigou. 2011. Sentiment analysis of social media content using n-gram graphs. In *Proceedings of the 3rd ACM SIGMM international workshop on Social media*. 9–14.

[3] Reid Andersen and Kevin J Lang. 2006. Communities from seed sets. In *Proceedings of the 15th international conference on World Wide Web*. 223–232.

[4] Renzo Angles and Claudio Gutierrez. 2008. Survey of graph database models. *ACM Computing Surveys (CSUR)* 40, 1 (2008), 1–39.

[5] Sabeur Aridhi and Engelbert Mephu Nguifo. 2016. Big graph mining: Frameworks and techniques. *Big Data Research* 6 (2016), 1–10.

[6] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. 2002. Models and Issues in Data Stream Systems. In *Proceedings of the Twenty-First ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS '02)*. Association for Computing Machinery, Madison, Wisconsin, 2. https://doi.org/10.1145/543613.543615

[7] Graham Cormode and Shan Muthukrishnan. 2005. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms* 55, 1 (2005), 58–75.

[8] Santo Fortunato. 2010. Community detection in graphs. *Physics reports* 486, 3-5 (Feb. 2010), 82–83. https://doi.org/10.1016/j.physrep.2009.11.002

[9] Alexandre Hollocou, Julien Maudet, Thomas Bonald, and Marc Lelarge. 2017. A linear streaming algorithm for community detection in very large networks. *CoRR* (2017). https://doi.org/10.48550/arXiv.1703.02955

[10] Wolfgang Huber, Vincent J Carey, Li Long, Seth Falcon, and Robert Gentleman. 2007. Graphs in molecular biology. *BMC bioinformatics* 8, 6 (2007), 1–14.

[11] Isa Inuwa-Dutse, Mark Liptrott, and Ioannis Korkontzelos. 2021. A multilevel clustering technique for community detection. *Neurocomputing* 441 (2021), 64–78.

[12] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

[13] Panagiotis Liakos, Alexandros Ntoulas, and Alex Delis. 2017. COEUS: community detection via seed-set expansion on graph streams. In *2017 IEEE International Conference on Big Data (Big Data)*. IEEE, 676–685.

[14] Fragkiskos D Malliaros and Michalis Vazirgiannis. 2013. Clustering and community detection in directed networks: A survey. *Physics reports* 533, 4 (2013), 95–142.

[15] Andrew McGregor. 2014. Graph stream algorithms: a survey. *ACM SIGMOD Record* 43, 1 (2014), 9–20.

[16] Justin J Miller. 2013. Graph database applications and concepts with Neo4j. In *Proceedings of the southern association for information systems conference, Atlanta, GA, USA*, Vol. 2324.

[17] Ferozuddin Riaz and Khidir M. Ali. 2011. Applications of Graph Theory in Computer Science. In *2011 Third International Conference on Computational Intelligence, Communication Systems and Networks*. 142–145. https://doi.org/10.1109/CICSyN.2011.40

[18] Satu Elisa Schaeffer. 2007. Graph clustering. *Computer science review* 1, 1 (2007), 27–64.

[19] Jonathan Shlomi, Peter Battaglia, and Jean-Roch Vlimant. 2020. Graph neural networks in particle physics. *Machine Learning: Science and Technology* 2, 2 (2020), 021001.

[20] Joyce Jiyoung Whang, David F Gleich, and Inderjit S Dhillon. 2013. Overlapping community detection using seed set expansion. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*. 2099–2108.

[21] Reda Yacouby and Dustin Axman. 2020. Probabilistic Extension of Precision, Recall, and F1 Score for More Thorough Evaluation of Classification Models. In *Proceedings of the First Workshop on Evaluation and Comparison of NLP Systems*. Association for Computational Linguistics, Online, 79–91. https://doi.org/10.18653/v1/2020.eval4nlp-1.9

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009