A Distributed Algorithm for Large-Scale Graph Clustering

Wissem Inoubli, Sabeur Aridhi, Haithem Mezni, Maddouri Mondher, Engelbert Nguifo

A Distributed Algorithm for Large-Scale Graph Clustering

Wissem Inoubli
LIPAH
Faculty of sciences of Tunis
Tunis, Tunisie
inoubliwissem@gmail.com

Sabeur Aridhi
University of Lorraine, LORIA
Vandoeuvre-lès-Nancy, France
sabeur.aridhi@loria.fr

Haithem Mezni
SMART Lab, Higher Institute of Management
Tunis, Tunisia
haithem.mezni@gmail.com

Maddouri Mondher
College Of Business, University of Jeddah
P.O.Box 80327, Jeddah 21589, Saudi Arabia
maddourimondher@yahoo.fr

Engelbert Mephu Nguifo
University of Clermont Auvergne, LIMOS
BP 10448, F-63000, Clermont-Ferrand, France
mephu@isima.fr

***Abstract*—Graph clustering is one of the key techniques to understand the structures present in the graph data. In addition to clusters detection, the identification of hubs and outliers is also a critical task as it plays an important role in the analysis of graph data. Recently, several graph clustering algorithms have been proposed and used in many application domains such as biological network analysis, recommendation systems and community detection. Most of these algorithms are based on the structural clustering algorithms SCAN. Yet, these algorithms have been designed for small graphs, without significant support to deal with big graphs. In this paper, we propose DSCAN, a novel distributed graph clustering algorithm based on SCAN. We present an implementation of DSCAN on top of BLADYG framework, and experimentally show the efficiency of DSCAN in the case of large networks.**

***Index Terms*—Big graph, Structural graph clustering, distributed computing, Big Data**

I. Introduction

Recently, the graph model has risen one of the most used data models in several applications like social networks [25], road maps [9], bioinformatics [33]. For instance, a recent ranking shows that the popularity of graph databases model increased up to around 500% in the last years [3], [19]. It has shown an optimum data model which allows to represent easily many relationships and to facilitate the exploration of data. Taking social networks as an example, the graph model organizes data elements into a set of vertices representing the members, and a list of edges to materialize the relationships between vertices. Also, the last years featured a big data explosion especially in graph-based social networks. As an example, Facebook in 2013 had over 874 million monthly users [6]. This proliferation of a huge amount of data and the massiveness of graphs introduce additional factors to the renewed popularity of graph analytics [19]. Consequently, new applications and use cases have been mentioned in the literature. One important analysis technique in the graph mining and graph anlysis fields is graph clustering. Graph clustering helps identify tightly connected regions within a graph [35]. It has been used to solve various problems such as discovering communities in social networks. In bioinformatics for instance, a group of closely connected proteins contributes to perfom a particular function in the body [33].

In some applications, the used graph could be very large. In these cases, the considered graph could be partitioned into several sub-graphs and the computation is perfeomed in a parallel/distributed way. Generally, graph clustering algorithms are used to ensure the graph partitioning [1], [34]. Likewise, detecting and analyzing research communities is a real world application which uses graph clustering to aggregate the authors under the same area according to the co-authorship (collaboration) relationships [2].

In this paper, we propose DSCAN: a novel graph clustering algorithm in the context of large graphs. Our main contributions in this work are summarized as follows:

- We propose a distributed graph clustering algorithm of large graphs. The proposed method is exact and allows discovering the same clusters that are discovred by SCAN, the referent algorithm for strctural graph clustering [32].
- We conduct an extensive experimental study with large graphs, to evaluate the scalability of DSCAN. We compare DSCAN with existing structural graph clustering algorithms.

The rest of the paper is organized as follows. After introducing graph clustering in Section I, we present a brief overview of related work in Section II. In Section III, we present the basic concepts related to the structural graph clustering. In Section IV, we present our proposed algorithm for large graph clustering. In Section V, we provide the experimental results. The last section is devoted to the conclusion and the future work.

II. Related work

Several graph clustering algorithms have been proposed in the literature. Taking as examples, modularity based approaches [21] that represent an optimzation problem which aims to optimize the modularity measure for each partitioning schema (generated randomly or according to a heurestic function) [21]. The Louvain method [5] represents one of the clustering algorithms based on modularity, that generates initially a random clustering, and after that, it starts to change every time a vertex from a cluster to another until getting a maximum modularity. Despite the fact that it gives very connected clusters in the larger graphs, the modularity measure cannot capture the small clusters [20]. Graph partitioning [7] and min-cut [14] are other methods used for the graph clustering which consist of splitting one graph into sub-graphs while optimizing the cut edge during the partitioning. The spectral clustering is another class of graph clustering algorithms [31] that is based on the graph density. It represents an input graph with a matrix and transforms this matrix to be able to apply the basic clustering algorithm like k-means [18]. Graph embedding has also been used in the graph clustering. Like in [17], the authors discussed the use of the graph embedding technique to combine the structural and attributed similarity over the graph clustering.

The above methods provide as output a list of clusters which are not really sufficient to understand the graph bevavior. To address this challenge, the Structural Clustering Algorithm for Networks (SCAN) was proposed in [32] aiming, not only to identify the clusters in the graph, but also to provide additional informations like hubs (vertices between one or more clusters) and outliers (vertices that do not belong to any cluster). These additional pieces of information can be used to detect vertices that can be considered as noise and also vertices that can be considered as bridges between clusters. The functional principle of SCAN is based on graph topology. It consists of grouping vertices that share the maximum number of neighbors. Moreover, it computes the similarity between all the edges of the graph in order to perform the clustering. The similarity computations step in SCAN is linear according to the number of edges, which degrades SCAN performance especially in case of large graphs. Structural graph clustering is one of the most effective clustering methods for differentiating the various types of vertices in a graph. In the litturature, several works have been proposed for the strucural graph clustering to overcome the drawback of SCAN. In [27], Shiokawa et al. proposed an extension of the basic SCAN algorithm, namely SCAN++. The proposed algorithm aims to take into account a property which can be considered as vertex and its two-hop-away vertices. SCAN++ could save many structural similarity operations, sinse it avoids several computing of structural similarity between vertices that are shared between the neighbors of a vertex and its two-hopaway vertices.

In the same way, the authors in [10] suppose that the identification of core vertices represents an essential and expensive task in SCAN. Based on this assumption, they proposed a pruning method for identifying the core vertices after a pruning step, which aims to avoid a high number of structural similarity computations. To improve the performance and robustness of the basic SCAN algorithm, an algorithm named LinkSCAN* has been proposed in [23]. LinkSCAN* is based on a sampling method, which is applied on the edges of a given graph. This sampling aims to reduce the number of structural similarity operations that should be executed. However, it provides approximate results.

Other works have proposed parallel implementations of SCAN algorithm [29] [24] [11] [28] [30]. In [29], the authors proposed an approach based on openMP library [8]. The author's aims were to ensure a parallel computation of the structural similarity and to show the impact of parallelism on the response time. Their method was proven to be faster than the basic SCAN algorithm. Other works have focused on the problem of dynamic graph clustering. In [24] and [11], the authors have extended SCAN algorithm to deal with the addition or removal of nodes and edges. Authors in [28], use the graphical processing unit (GPU) whose purpose is to parallelize the processing, and to benefit from the high number of processing slots in the GPU which increase the degree of parallelism. Most of the above presented works suffer from two major problems: (1) they do not deal with big graphs and (2) they do not consider already distributed/partioned graphs. Through Tab I, we have summarized the discussed approaches according to some features. As shown in Tab. I, most proposed algorithms allow parallel processing but could not deal with very large graphs. It is also clear that none of the studied approaches allow distributed computing. In this work, we tackle the problem of large graph clustering in a distributed setting.

III. Background

A. Structural graph clustering: Basic concepts

Graph clustering consists in dividing a graph into several partitions or sub-graphs. As with other clustering techniques, we must use one or more metrics to measure the similarity between two vertices or partitions in the graph. In the structural clustering technique, the graph structure or topology is splitted into a set of subgraphs that are relatively distant and a set of vertices strongly connected.

As a well-known algorithm for structural graph clustering, SCAN algorithm uses the structural similarity between vertices to perform the clustering. In addition, it provides other pieces of information like outliers and bridges. In what follows, we first present an overview of graphs and graph clustering with the SCAN algorithm.
Consider a graph $G = \{V, E\}$, where V is a set of vertices, and E is a set of edges between vertices. Each of those elements can represent a real property in real-word applications. Let u and v two vertices in V, we denote (u,v) an edge between u and v; u (resp. v) is said to be a neighbor of v (resp. u).
In the following, we extend some basic definitions of structural graph clustering, which will be used in our proposed algorithm.

Definition 3.1: (Structural neighborhood) The structural neighborhood of a vertex v, denoted by $N(v)$, and represents

TABLE I: Comparative study on existing graph clustering methods

Approach	**Parallel**	**Distributed**	**Processing model**	**Graph partitioning**	**Main contributions**
SCAN [32]	No	No	Sequential	No	basic implementation of structural graph clustering
SCAN++ [27]	No	No	Sequential	No	reducing the number of similarity computations
AnySCAN [36]	Yes	No	Parallel	No	parallelizing SCAN
pSCAN [10]	Yes	No	Parallel	No	reducing the number of similarity computations
Index-based SCAN [30]	No	No	Sequential	No	interactive SCAN
ppSCAN [12]	Yes	No	Parallel	No	parallel version of pSCAN
SCAN based on GPU [28]	Yes	No	Parallel	No	a GPU-based version of SCAN
pm-SCAN [26]	Yes	No	Parallel	Yes	graph partitioning to reduce the I/O costs

all the neighbors of a given vertex $v \in V$ including the vertex v:

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

Definition 3.2: (Structural similarity) The structural similarity between each pair of vertices (u, v) in E represents a number of shared structural neighbors between u and v. It is defined by $\sigma(u, v)$.

$$\sigma(u,v) = \frac{|N(u)| \cap |N(v)|}{\sqrt{|N(u)| \,.\, |N(v)|}} \tag{2}$$

After calculating the structural similarity with Eq. (2), SCAN uses two parameters to detect the core vertices in a given graph G.

Definition 3.3: (ϵ-neighborhood) Each vertex has a set of structural neighbors, like it is mentioned in Definition 3.1. To group one vertex and its neighbors in the same cluster, they must have a strong connection between them. These strong connections denoted by ϵ-neighborhood. SCAN uses a ϵ threshold and the Eq. 3 to filter, for each vertex, its strongest connections. The ϵ-neighborhood is defined as follows.

$$N_\epsilon(u) = \{N(u) | \sigma(u,v) \geq \epsilon\} \tag{3}$$

Note that the ϵ threshold $0 < \epsilon \leqslant 1$ shows to what extent two vertices u and v are connected based on the shared structural neighbors. In addition, it represents a metric with which the most important vertices, also called *core* vertices, are detected.

Definition 3.4: (Core) Core vertices detection is a fundamental step in SCAN algorithm. It consists of finding the dominant vertices in a given graph G. This step allows to build the set of clusters or a clustering mapping. A core vertex v is a vertex who has a sufficient number of neighbors strongly connected with it $N_\epsilon(v)$. We use μ as a minimum number of strong connected neighbors (see Definition 3.3). A core vertex $core$ is modeled as follows:

$$V_c = \{v \mid N_\epsilon(v) \geq \mu\} \tag{4}$$

Definition 3.5: (Border) Let v_c is a core vertex, v_c has two lists of structural neighbors: (1) weak connected neighbors to v_c, also called noise vertices $(N(V_c) \setminus N_\epsilon(V_c))$, and (2) strong connected neighbors called *reachable structured neighbors*. In our work, reachable structured neighbors are called border vertices. $N_\epsilon(V_c)$ represents the border vertices of a core vertex V_c.

Once the nodes and their borders are determined, it is straightforward to start a clustering step. To do so, we use the following definition:

Definition 3.6: (Cluster) A cluster C ($|C| \geq 1$) is a nonempty subset of vertices, where its construction is based on the set of core vertices and their border vertices. The clusters' building scenario is as follow: first, randomly chose a core from the cores' list and create a cluster C, then push the core and its borders into the same cluster. At the same time, we must check if the list of borders has a core vertex. We must insert their borders into the same cluster and we apply this step recursively until adding all the borders of the connected cores. Finally, we chose other core vertices and we apply the same previous step until checking all core vertices.

Among the fundamental informations returned by SCAN, compared to others algorithms for graph clustering, we mention hub and outlier informations, defined as below:

Definition 3.7: (Bridge and Outlier) The clustering step aggregates the core vertices and their borders into a set of clusters. However, some vertices do not have strong connections with a core vertex, which does not give the possibility to rejoin any cluster. In this context, SCAN algorithm classifies these vertices on two families bridges and outliers.

A vertex v, that is not part of any cluster and has at least two neighbors in different clusters, is called a bridge vertex. Otherwise it is considered an outlier.

B. Running example

In this section, we explain through a running example, how SCAN algorithm works. Consider a graph G presented in Figure 1 and the parameters $\epsilon = 0.7$ and $\mu = 3$. In the first step, line 1-3 of Algorithm 17 use Eq. 2 to compute the structural similarity for each edge $e \in \mathbb{E}$. Then, it uses Eq. 3 (line 5-8) to define the core vertices (see gray vertices in Fig.1). After that, we proceed to the clustering step, then we apply Eq. 3.6 (line 11-16) to build the clustering schema. In our example, we have four core vertices: 0,2,9 and 11. Each core has a list of border vertices (the neighbors which have strong connections with a core). For example: in our case, we have a vertex number 2 as a core and it has the vertices number 1, 4, 5, 3 and 0 as list of borders since they have strong

Algorithm 1: Basic SCAN algorithm

```
Input  : A Graph $G$ and parameters ($\mu$ ,$\epsilon$)
Output: $\mathbb{C}, \mathbb{B}, \mathbb{O}$
foreach $(u, v) \in E$ do
    Compute $\sigma(u, v)$
end
$Core \leftarrow \emptyset$
foreach $u \in V$ do
    if $(\mid N_\epsilon(u) \geq \mu)$ then ;
    $Core = Core \cup \{u\}$
end
$Cluster \leftarrow \emptyset$
foreach unprocessed core vertex $u \in Core$ do
    $Cluster$= $\leftarrow \{u\}$
    Mark $u$ processed
    foreach unprocessed border of vertex $v \in N_\epsilon(u)$ do
        $Cluster$= $\leftarrow$Cluster $\cup \{v\}$
        Mark $v$ processed
    end
end
```

connections with the core. The core and its borders build a cluster, and if a border is a core we joint all its borders into the cluster. Like in our example, the vertex 2 is a core. Hence, we joint all its borders (1, 3 ,5 ,4 including 0 as being a core vertex). In this case, if the vertex 3 is not a border of vertex 2 and it is a border of vertex 0, then vetrex 3 should belong to the same cluster of vertex 2 which is a core vertex, since it is a reachable border of vertex 2. The last step of SCAN algorithm consists in defining the bridges and the outliers. As shown in Fig. 1, we have two clusters. The first one is composed of vertices 1,2,3,4, and 5. The second cluster is composed of nodes 8,9,10, and 11. The remaining vertices (6 and 7) must be categorized as outliers and bridges according to the Definition 3.7. In our example, vertex 7 has two connections with two different clusters and vertex 6 has only one connection with one cluster which makes vertex 7 a bridge and vertex 6 an outlier.

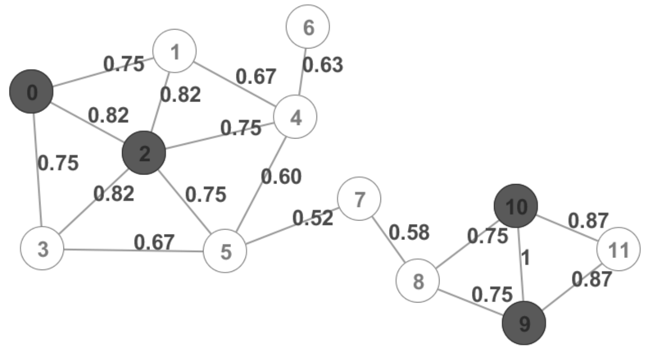


Fig. 1: Running example ($\epsilon = 0.7$ and $\mu = 3$).

IV. DSCAN: A Distributed Algorithm for Large-Scale Graph Clustering

In this section, we introduce DSCAN, a new distributed algorithm for structural graph clustering. Our proposed approach is based on a master/slave architecture and is implemented on top of BLADYG framework [4], a distributed graph processing framework in which the slaves are responsible for the execution of a specific computation and the master machine coordinates between all the slaves. The input data must be divided into sets of chunks (sub-graphs in our case). Each chunck/partition is assigned to a worker, which performs a specific computation. The master machine orchestrates the execution of the workers. In the following, we present the main two steps of DSCAN: (1) graph parittioning and (2) graph clustering.

A. Distributed graph partitioning

In this step, we split the input graph G into several small partitions $P_1P_2..P_n$ while keeping data consistency (graph structure). To ensure the consistency property while dividing the input graph, we must identify a list of cuts edges in order to have a global view of G. Usually, the graph partitioning problem is categorized under the family of NP-hard problems, that need to parse all the combinations in order to have the best partitioning result [13]. For this reason, we proposed an approximation and a distributed partitioning algorithm as a preliminary step for our distributed clustering algorithm. Algorithm 2 shows that, at the beginning, the master machine divides equitably an input graph file into sub-files according to the number of edges, and sends the sub-files to all the workers. Secondly, each worker gets a list of edges and vertices from its sub-file. Thereafter, it sends its list of vertices to all workers, in order to determine the frontier vertices so that to get the cuts edges. Afterwards, when each worker receives a list of vertices from its neighbor workers, it determines the vertices that belong to the current worker (partition). Consequently, these vertices are considered as frontier vertices in their partitions. In the last step, when each worker could determine the frontier vertices, it starts to fix the cuts edges i.e. edges that have a frontier vertex.

B. Distributed clustering

1) Algorithm: As shown in Algorithm 3, DSCAN has two main steps: (1) local clustering step and (2) merging step.

Step 1: Local clustering. Like it is presented in Algorithm 3, the input graph G is splitted into multiple sub-graphs/partition ($\mathbb{P}$), each one is assigned to a worker machine. The partitioning step as mentioned in Section IV-A is performed according to the α parameter, which refers to the number of worker machines (line 1). To overcome the loss of information during the partitioning step (edges connecting nodes in different partitions), frontier vertices are duplicated into neighboring partitions (line 5-8). Subsequently, for each partition P_i, a local clustering is performed (lines 9-14) on each worker machine. Fig. 2 shows a demonstrative example of the duplication step. The demonstrative example describes how

Algorithm 2: Distributed partitioning algorithm

```
Input  : Graph file $GF$ as a text file, parameter ($NP$
         number of partitions)
Output: $\mathbb{P}$ set of partitions
BLADYG initialization according to the $NP$ parameter ;
Master machine: split $GF$ into a set of sub-files $GF$ ;
foreach $GF_i \in \mathbb{GF}$ do
    Assign $GF_i$ to $W_i$ ;
end
/* In parallel */
foreach $Worker \in \mathbb{W}$ do
    Get list of vertices and edges from $GF_i$;
    Find the list of frontier vertices from the neighboring
      workers;
end
/* In parallel */
foreach $Worker \in \mathbb{W}$ do
    foreach $Edge \in \mathbb{E}$ do
        (a,b)=Edge ;
        Let $P_i^f$ the frontier vertices;
        if ($a \in P_i^f \cup b \in P_i^f$ ) then ;
        Set the edge $E$ as a cut edge;
    end
end
```

the consistency of graph will be ensured during the partitioning step.

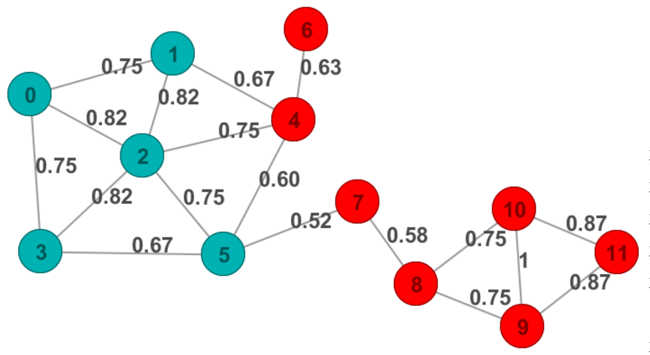


Fig. 2: Illustrative example

Assume that, a graph G is partitioned into tow partitions $P1$ (vertices in blue) and $P2$ (vertices in red), like it is depicted in Fig. 2, and each partition has a set of vertices connected with other partitions. We call them frontier vertices of a given partition P, and denoted by V_P^f. For example, V_{P1}^f={1,2,3} and V_{P2}^f={4,7}. Each $v \in V_P^f$ has a set of internal neighbors and external neighbors, (e.g., vertex 4 is a vertex of the partition $P2$, it has vertex 6 as internal and vertices 1,2 and 5 as external neighbors). Computing the structural similarity of a frontier vertex u considers that all neighbors of u belong to the same partition. Thereby, we duplicate all frontier vertices in partition P to all neighboring partitions and we

Algorithm 3: DSCAN

```
Input  : Graph $G$ , parameters ($\mu, \epsilon, \alpha$)
Output: $\mathbb{C}, \mathbb{B}, \mathbb{O}$
/* Divide G into sub-graphs $\mathbb{G} = \{G_1 G_2 .. G_\alpha\}$
   according to parameter $\alpha$ */
$\mathbb{P} \longleftarrow$ Partition ($G$,$\alpha$) ;
/* In parallel */
foreach $P_i \in \mathbb{P}$ do
    Assign $P_i$ to $W_i$ ;
end
/* In parallel */
/* Step 1: Local clustering */
foreach $Worker \in \mathbb{W}$ do
    Let $P_i$ the current partition ;
    Find the frontier vertices in $P_i$ and duplicate them into
      neighboring partitions ;
end
/* In parallel */
foreach $Worker \in \mathbb{W}$ do
    Compute the structural similarity of a partition $P_i$ using
      $V^f$ list;
    Retrive local Cores and Borders in $P_i$ ;
    Build local clusters in $P_i$;
    Find local Bridges and Outliers in $P_i$;
end
/* Step 2: Merging */
All workers send theirs local clusters between them: using
Worker2Worker message;
foreach $Worker \in \mathbb{W}$ do
    if ($C_1 \cap C_2 \cap .. \cap C_\alpha = \mathbb{V}$; and $\exists V_i \in \mathbb{C}ore$ according to
      4.1) then ;
    $C \leftarrow$ Mergre($C_1, C_2, ..C_\alpha$);
    Send $C$ to master;
    else
        Send local clusters to master;
    end
    for $V_i$ IN $\mathbb{O}utliers$ do
        if ($V_i \in \mathbb{C}ore \cup \mathbb{B}order \cup \mathbb{B}ridges$ according to 4.2 )
          then ;
        Remove $V_i$ from the list of $\mathbb{O}uliers$;
    end
    for $V_i$ IN $\mathbb{O}utliers$ do
        $Nb_Connections \leftarrow 0$;
        for $C_i$ IN $\mathbb{C}lusters$ do
            if ($N(V_i) \cap C \neq \varnothing$) then ;
            $Nb_Connections++$;
        end
        if ( $Nb_Connections \geq 2$ according to 3.7 ) then ;
        Add $V_i$ to $\mathbb{B}ridges$ ;
         Remove $V_i$ from $\mathbb{O}utliers$;
    end
end
Send C,B,O to master using Worker2Master message;
```

call them external vertices. For example in our demonstrative example: $P1$ has frontier vertices V_{P1}^{f}={1,2,3} and $P2$ is the neighboring partition. Thus, we must duplicate V_{P1}^{f} into $P2$ to ensure the accuracy of structural similarity of V_{P2}^{f} (see Fig. 3).

After that, when we apply a local clustering on $\mathbb{P}1$ and $\mathbb{P}2$, we will find several conflicts such as the vertex $v \in V_{P1}^{f}$ is a core vertex in $\mathbb{P}1$, and an outlier in $\mathbb{P}2$. These conflicts should be avoided in the merging step.

Step 2: Merging. The distribution of similarity computation and the local clustering step can improve the response time of our proposed DSCAN, compared to the basic sequential algorithm. However, we should take into consideration the exactness of the returned results compared to those of the basic SCAN. To ensure the same result of basic SCAN, we define a set of scenarios to ensure the merging step. These scenarios will repetitively be applied to to every two partitions of G until combining all the partitions (see Algorithm 3 from line 16 to line 37). For each pair of partitions P_i and P_j, a merging function is executed to combine the local results from P_i and P_j and store them in global variables like clusters, borders, bridges and outliers. Algorithm 3 also presents several scenarios (Lemmas 4.1, 4.2 and 4.3) to solve encountered conflicts mentioned below.

Lemma 4.1: **(Merging local clusters)** Let $\mathbb{C}_1$ and $\mathbb{C}_2$ two sets of local clusters in different partitions P_1 and P_2, respectively. $\exists c_1 \in \mathbb{C}_1$ and $\exists c_2 \in \mathbb{C}_2$, $Core(c_1) \cap Core(c_2) \neq \emptyset$.
Let C_i be a cluster that groups a set of border and core vertices. If C_i shares at least one core vertex c with another cluster C_j, then c has a set of borders in C_i and C_j, and all the vertices in C_i and C_j are reachable from c. Hence, C_i and C_j should be merged in the same cluster.

Lemma 4.2: **(Outlier to Bridge)** Let $\mathbb{C}_1$ and $\mathbb{C}_2$ two sets of local clusters in both partitions P_1 and P_2, respectively.
$\exists$ C_1 and C_2 two clusters belong to the two different sets of clusters $\mathbb{C}_1$ and $\mathbb{C}_2$. In addition, $\exists$ o an outlier in both partitions P_1 and P_2 and $N(o) \cap c_1 \neq \emptyset$ and $N(o) \cap c_2 \neq \emptyset$

If C_i and C_j $(i \neq j)$ share an outlier o, this means that o is weak connected with two clusters C_i and C_j. Hence, according to the Definition 3.7, o should be changed to bridge vertex.

Lemma 4.3: **(Bridge to Outlier)** Let C_1 and C_2 two local clusters in both partitions P_1 and P_2, respectively.
$\exists$ a bridge b according to only tow clusters c_1 and c_2, when c_1 and c_2 two clusters will be merged into one cluster (during the merging step), b should be changed into an outlier.

Let C_i and C_j two clusters that have a set of vertices (borders or cores), and have a set of bridges with other local clusters, and $\exists$ b a bridge vertex according only to the two clusters C_i and C_j where $i \neq j$. In the merging step and according to Lemma 4.1, if one or several clusters share at least a core vertex, thus we merge them into the same cluster. In this case (C_i,C_j) $\Rightarrow$ C what make b be weak connected with only one cluster (C), then according to Definition 3.7, b should change its status from bridge to outilier.

For instance from lines 16 to 22, we have focused on the shared cores between two clusters and the case when they share at least one core. According to Lemma 4.1, we should merge them into the same cluster. Subsequently, in lines 23-25, we verify for each outlier if it does not belong to some sets of cores, borders or bridges. In this case, we must remove it from the list of outliers. Otherwise, a vertex v should be changed as a bridge if it is classified as an outlier in two clusters C_i and C_j that are not in the same partition, and if it has two connections with different clusters in the merging step.

2) Illustrative example: Fig. 3 shows a demonstartive example of a graph clustering using DSCAN, in this example we use the same parameters (ϵ and μ) of the running example in Section III, and two partitions $P1$ and $P1$. In the first step of DSCAN, the input graph G is divided into two partitions $P1$ and $P1$ as it presented in Fig. 3 where the blue vertices represent the first partition and the red vertices represent the second partition. In the second step, DSCAN duplicates the frontier vertices in each pair of adjacent partitions. For example, in our example the blue vertices 1,2 and 4 are duplicated in partition $P2$ since they represent frontier vertices in their partition. In the same way, the vertices 4 and 7 are duplicated in the partition $P2$. In the third step, all workers perfom the similarity computation, check the status for each vertex and build the local clusters as it shown in Fig 3. The last step of DSCAN consists of combining all local results returned by each worker. As shown in Fig 3, there are some conflicts in terms of node status. For example, vertex 2 is a core vertex in $P1$ whereas it is a noise (outlier) vertex in $P2$. In the same way, vertex 4 is a border in $P1$ and a noise in $P2$. Then, after building the local clusters, $P1$ has one cluster (1,2,3,4, and 5) in which vertex 7 is a noise vertex in $P2$. In the same way, $P2$ groups the vertices 8,9, 10 and 11 as a cluster and the remaining vertices (4,5,1,2,7 and 6) are considered as noise vertices including vertex 7 (which is a bridge according to basic SCAN (see running example in Section III)), In the merging step, DSCAN considers that the vertex 7 has two weak connections with two different clusters, thus vertex 7 is marked as a bridge.

V. Experiments

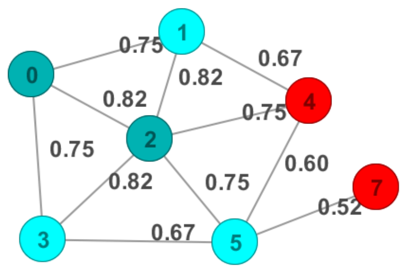
In this section, we present an experimental study and we evaluate the effectiveness and efficiency of our proposed algorithm for structural clustering of large distributed graphs.

A. Experimental protocol

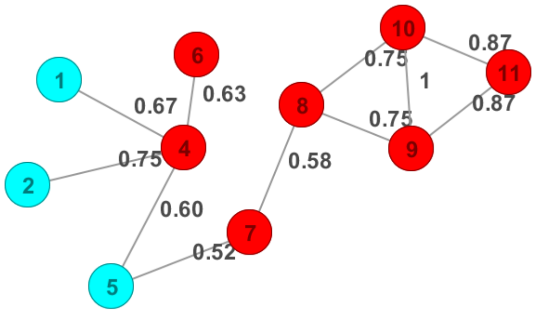
We compared DSCAN with four existing structural graph clustering algorithms:

1) Basic SCAN [1].
2) pSCAN: a pruning SCAN algorithm[2].
3) AnyScan: a parallel implementation of basic SCAN using OpenMP library[3].

[1]https://github.com/eXascaleInfolab/pSCANdeploymet
[2]https://github.com/RapidsAtHKUST/ppSCAN/tree/master/SCANVariants/anySCAN
[3]https://github.com/RapidsAtHKUST/ppSCAN/tree/master/SCANVariants/anySCAN



(a) Partition 1



(b) Partition 2

Fig. 3: The graph G (used in the running example in Section III) partionned into two partitions $P1$ and $P2$

4) ppSCAN: a pruning and parallel SCAN implementation [4].

The above mentioned algorithms are implemented with C language. Thus, we used the GCC/GNU compiler to build their binary versions. The compared algorithms are divided into two categories: (1) centralized algorithms such as SCAN and pSCAN, and (2) parallel algorithms such as AnyScan and ppSCAN. To run both centralized and parallel algorithms, we used a *T3.2xlarge* virtual machine on Amazon EC2. This machine is equiped with an 8 vCPU Intel Skylake CPUs at 2.5 GHz and 32 GB of main memory on a Ubuntu 16.O4 server distribution. In order to evaluate DSCAN, we used a cluster of 10 machines, each of them is equipped with a 4Ghz CPU, 8 GB of main memory and operating with Linux Ubuntu 16.04.

B. *Experimental data*

For all test cases, we used real-world graph datasets (see Table II) obtained from the Stanford Network Analysis Project (SNAP) [22].

C. *BLADYG Framework*

DSCAN is implemented on top of the BLADYG framework. Algorithms 2 and 3 are defined according to the BLADYG architecture. Thereby in the following, we present some details about the used framework. BLADYG is a distributed and parallel graph processing framework that runs on a commodity hardware. The architecture of BLADYG is based on a mster/slaves topology. BLADYG starts to read the input graph from many different sources, which can be local or distributed files such as Hadoop Distributed File System (HDFS) and Amazon Simple Storage Service (Amazon S3). The communication model used by BLADYG is the message passing technique which consists in sending messages explicitly from one component to another in order to get or send some data during the processing. In the same way, BLADYG defines two types of messages: (1) worker-to-worker messages, and (2) master-to-worker messages. BLADYG allows its users to implement their partitioning techniques.

[4]https://github.com/RapidsAtHKUST/ppSCAN/tree/master/ppSCAN-release

D. *Experimental results*

Speedup. We evaluated the speedup of DSCAN compared to the basic SCAN and its variants presented in Section V-A. The compared algorithms use different graph representations. In fact, AnyScan and SCAN implementations use the adjacency list representation [15] whereas, both pSCAN and ppSCAN use the Compressed Sparse Row (CSR) format [16]. In our proposed algorithm, we used an edge list format, in which each line represents one edge of the graph. In fact, the incompatibility of the graph representations poses an additional transformation cost while evaluating the studied methods. For example, the transformation of the live journal dataset from edge list to adjacency list takes around 100 seconds using one machine equiped with an 8 vCPU and 32 GB of main memory.

Fig. 4 shows the runtime of the studied approaches with different datasets.

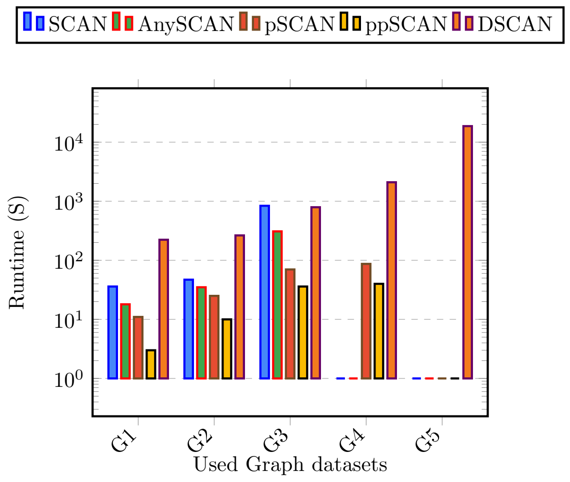


Fig. 4: Impact of the graph size on the processing time of both SCAN and DSCAN

As shown in Fig. 4, our approach is slower than the other algorithms in the case of small graphs (G1,G2,G3) and there was a very large gap between DSCAN and the other algorithms

TABLE II: Graph datasets

Dataset name	Number of vertices	Number of edges	Diameter	Avg. CC
G1: California road network	1 965 206	2 766 607	849	0.04
G2: Youtube	1 134 890	2 987 624	20	0.08
G3: Orkut	3 072 441	117 185 083	9	0.16
G4: LiveJournal	3 997 962	34 681 189	17	0.28
G5: Friendster	65 608 366	1 806 067 135	32	0.16

especially with pSCAN and ppSCAN. On the other hand, this gap starts to be reduced when the graph size increases (case of G4 dataset). The plots bar in Fig. 4 shows a gap of 12x between DSCAN and basic SCAN with the G1 dataset and 2x only with the G4 dataset. We notice that the gap between DSCAN and ppSCAN depends mainly on the size of the used dataset. For example, with the G1 dataset, the gap between DSCAN and ppScan is about 20x, while this gap is reduced to 11x for the G4 dataset. This can be explained by several reasons such as the pruning step of pSCAN, which exempts several similarity computings during the clustering step. It is important to mention that DSCAN is a distributed implementation of SCAN and the other studied algorithms are centralized. This leads to additional costs related to data distribution, synchronization and communication. In addition, Fig 4 shows that with the modest hardware configurations, only DSCAN can scale with large graphs like the G5 dataset.

Scalability. The main goal of this experiment is to evaluate the horizontal scalability of our algorithm. We used two graphs (LiveJournal and California road network). We set the values of μ and ϵ to 3 and 0.5 respectively, and we varied the number of machines, with the goal to measure the response time in each size of the cluster. It can be clearly seen, from Fig.5, that our algorithm is horizontally scalable, which was not guaranteed by the other algorithms, as discussed in the state-of-the-art section. Fig.5 also shows that the running time will decrease depending on the number of machines in the cluster. When we add a new machine to the cluster, the running time becomes smaller. As depicted in Fig. 5, the red curve (LiveJournal graph) shows a significant improvement in response time of about 82% when the number of machines reaches 10. We also notice a weak improvement, according to the number of machines in the cluster, when we have a small graph (case of California road network which is smaller than LiveJournal graph). The red curve's behavior can be explained by the splitting of the input graph into small sub-graphs and by performing a local clustering on each sub-graph, which reduces the global response time even with the additional cost of aggregating intermediate results returned by each machine of the cluster. That was not the case with blue curve, where we have an improvement of about 50% using a cluster of 10 machines, compared to the results using a 6-machine cluster. We note that the curve starts to increase when the cluster size exceeds 8 machines. This is due to the communication in the shuffling step.

Impact of ϵ value on DSCAN. The number of clusters, bridges and noise vertices depends on the values of ϵ and μ. In ppSCAN, when we decrese the value of μ the running time increases, as the non-pruned edges are inreased. Similarly, in this experiment we evaluated the impact of ϵ value on the running time of DSCAN. For this, we varied the value of ϵ from 0.2 to 0.8. As shown in Fig. 6, the response time is slightly dependent on the value of ϵ, especially with large graphs. When we increase the value of ϵ the response time increases. Overall, the observed behavior can be explained by the merging step in DSCAN algorithm, since in the previous steps (graph loading and local clustering) we do not see the impact of ϵ on these steps. In the graph loading step, we do not use this value and in local clustering we use the basic SCAN clustering which is not dependent to ϵ. That is why, the impact of ϵ on the running time can by explaind by the merging step.

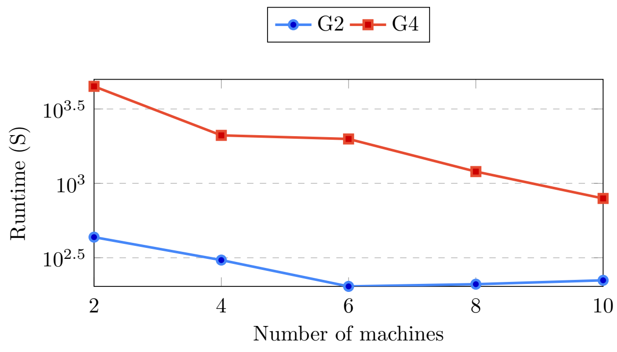


Fig. 5: Impact of the number of workers on the running time (with $\epsilon = 0.5$, $\mu = 3$).

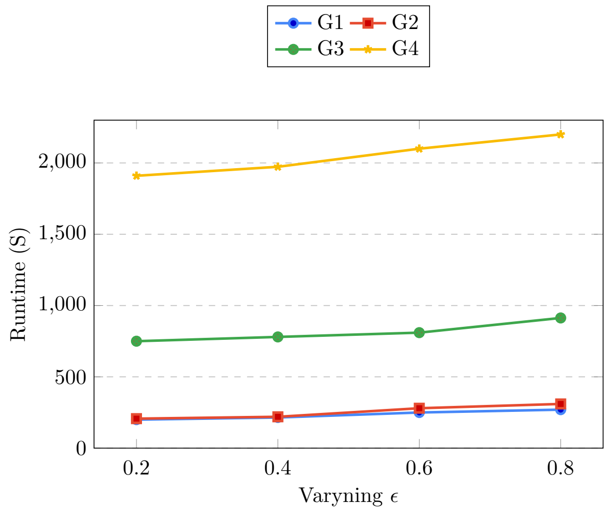


Fig. 6: Impact of the value of ϵ size on the running time of DSCAN

In fact, when the value of ϵ increases, the number of outliers becomes larger. Also, in the merging function (see Algorithm 3, lines 17-37), DSCAN combines the local results by starting to check the clusters that share almost one core to merge them. Then, it verifies for all outliers if they are bridges or cores. Hence, outliers' checking requires more communication between the workers, which increases the respense time.

Evaluation of DSCAN steps: DSCAN algorithm is based on four main steps. In each step, DSCAN performs a specific treatment with different costs in terms of running time. To study the running cost of each step in DSCAN, we used four graph datasets and we fixed the values of ϵ and μ to 0.5 and 3 respectively. Fig. 7 shows the response time of each step

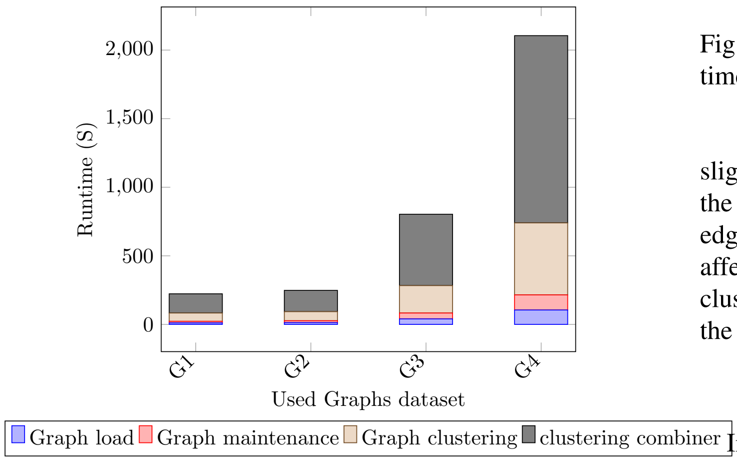


Fig. 7: Performance of DSCAN phases

in DSCAN. The merging function is the most expensive step that makes DSCAN slow, compared to the other algorithms. It takes more than 50% of the global running time with all the used graph datasets, while the clustering step takes about 30% only. The rest of computation time is devoted to the graph loading step and the duplication of vertices in frontiers to ensure the consistency property, as we discussed in Section IV-A. This disparity can be explained by the communication between machines during the aggregation of local results returned by each machine. Consequently, communication in DSCAN must be improved because the clustering step takes a little time which can make DSCAN faster.

Imapct of the graph partitioning on DSCAN. In our vision, the partitioning step has a direct impact on the response time of DSCAN. For this reason, we randomly generated four partitioning schemas, and for each one, we got the number of cut-edges as follows: 17,6M, 19,2M, 22,3M and 24,6M for the partitions P1, P2, P3 and P4 respectively. Then, we run DSCAN on all partitions with the same configuration (10 machines and ϵ=0.5 μ=3). As shown in Fig. 8, there is a very clear impact of the graph partitioning on the response time of DSCAN, as this latter rises from nearly 800 to 1000 seconds with P1 and P2. Furthermore, the elapsed time of each DSCAN step varies from one partitioning to another. This disparity is noticed mostly during the merging step and

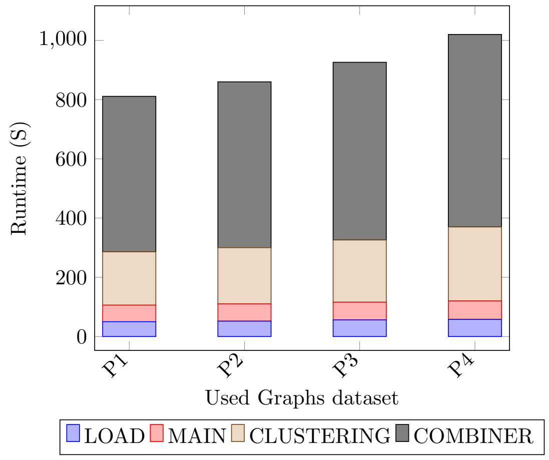


Fig. 8: Impact of the partitionning step on DSCAN response time (ϵ=0.5, μ=3)

slightly in the clustering step. This is probably explained by the number of vertices duplicated due to the number of cuts-edges produced by the graph partitioning. This number would affect the amount of similarity computing operations in the clustering step, and increases the communication cost during the merging step.

VI. Conclusion

In this paper, we proposed DSCAN, a distributed algorithm for big graph clustering based on the structural similarity. DSCAN is based on a distributed and master/slaves architecture which makes it scalable and works on the community of modest machines. The proposed algorithm is able to perform any size of the graph and can be scalable at a large number of machines in parallel ways. Starting, we have presented the principal function of DSCAN from the partitioning to the combiner of intermediate results for each worker. Also, we have performed an extensive experimentation about our proposed algorithm compared with other algorithms. The experiments have shown that DSCAN featured an horizontal scalability that is not guaranteed with other algorithms. In future works, we will firsttry to improve the graph partitioning step of DSCAN. Then, we will tackle the problem of clustering of large and dynamic graphs. In the partitioning step, we plan to use the density feature during the graph partitoning, whereas for the dynamic graph clustering, we plan to define strategies in order to deal with graph updates without recomputing the graph clustering from the scratch.

Acknowledgment

To the University of Clermont Auvergne and the LIMOS laboratory , we would like to express our sincere gratitude for providing human conditions, logistical and material support that helped us to the developement and conclusion of this project . Especially, for providing us the opportunity to perform the experiment part of this work in a data center called GALACTICA (https://galactica.isima.fr). We acknowledge also the financial support of our research study of the Tunisian government and particularly the Ministry of Higher Education throught the PHD scholarship for the academic year 2018-2019.

References

[1] Z. Abbas, V. Kalavri, P. Carbone, and V. Vlassov. Streaming graph partitioning: an experimental study. *Proceedings of the VLDB Endowment*, 11(11):1590–1603, 2018.

[2] T. Amjad, A. Daud, S. Khan, R. A. Abbasi, and F. Imran. Prediction of rising stars from pakistani research communities. In *2018 14th International Conference on Emerging Technologies (ICET)*, pages 1–6. IEEE, 2018.

[3] P. Andlinger. Graph dbms increased their popularity by 500% within the last 2 years, 2015.

[4] S. Aridhi, A. Montresor, and Y. Velegrakis. Bladyg: A graph processing framework for large dynamic graphs. *Big Data Research*, 9:9–17, 2017.

[5] T. Aynaud and J.-L. Guillaume. Static community detection algorithms for evolving networks. In *8th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks*, pages 513–519. IEEE, 2010.

[6] M. Baborska-Narozny, E. Stirling, and F. Stevenson. Exploring the relationship between a'facebook group'and face-to-face interactions in'weak-tie'residential communities. In *Proceedings of the 7th 2016 International Conference on Social Media & Society*, page 17. ACM, 2016.

[7] U. Brandes, M. Gaertler, and D. Wagner. Experiments on graph clustering algorithms. In *European Symposium on Algorithms*, pages 568–579. Springer, 2003.

[8] J. M. Bull. Measuring synchronisation and scheduling overheads in openmp. In *Proceedings of First European Workshop on OpenMP*, volume 8, page 49. Citeseer, 1999.

[9] L. Cao and J. Krumm. From gps traces to a routable road map. In *Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems*, pages 3–12. ACM, 2009.

[10] L. Chang, W. Li, L. Qin, W. Zhang, and S. Yang. pscan: Fast and exact structural graph clustering. *IEEE Transactions on Knowledge and Data Engineering*, 29(2):387–401, 2017.

[11] L. Chang, W. Li, L. Qin, W. Zhang, and S. Yang. pscan: Fast and exact structural graph clustering. *IEEE Transactions on Knowledge and Data Engineering*, 29(2):387–401, 2017.

[12] Y. Che, S. Sun, and Q. Luo. Parallelizing pruning-based graph structural clustering. In *Proceedings of the 47th International Conference on Parallel Processing, ICPP 2018, Eugene, OR, USA, August 13-16, 2018*, pages 77:1–77:10, 2018.

[13] I. S. Dhillon. Co-clustering documents and words using bipartite spectral graph partitioning. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 269–274. ACM, 2001.

[14] C. H. Ding, X. He, H. Zha, M. Gu, and H. D. Simon. A min-max cut algorithm for graph partitioning and data clustering. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 107–114. IEEE, 2001.

[15] B. Doerr and D. Johannsen. Adjacency list matchings: an ideal genotype for cycle covers. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1203–1210. ACM, 2007.

[16] E. F. DAzevedo, M. R. Fahey, and R. T. Mills. Vectorized sparse matrix multiply for compressed row storage format. In *International Conference on Computational Science*, pages 99–106. Springer, 2005.

[17] P. Goyal and E. Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, 2018.

[18] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[19] A. P. Iyer, A. Panda, S. Venkataraman, M. Chowdhury, A. Akella, S. Shenker, and I. Stoica. Bridging the gap: towards approximate graph analytics. In *Proceedings of the 1st ACM SIGMOD Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA)*, page 10. ACM, 2018.

[20] Y. Kozawa, T. Amagasa, and H. Kitagawa. Gpu-accelerated graph clustering via parallel label propagation. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 567–576. ACM, 2017.

[21] D. LaSalle and G. Karypis. Multi-threaded modularity based graph clustering using the multilevel paradigm. *Journal of Parallel and Distributed Computing*, 76:66–80, 2015.

[22] J. Leskovec and A. Krevl. {SNAP Datasets}:{Stanford} large network dataset collection. 2015.

[23] S. Lim, S. Ryu, S. Kwon, K. Jung, and J.-G. Lee. Linkscan*: Overlapping community detection using the link-space transformation. In *2014 IEEE 30th International Conference on Data Engineering (ICDE)*, pages 292–303. IEEE, 2014.

[24] S. T. Mai, M. S. Dieu, I. Assent, J. Jacobsen, J. Kristensen, and M. Birk. Scalable and interactive graph clustering algorithm on multicore cpus. In *Data Engineering (ICDE), 2017 IEEE 33rd International Conference on*, pages 349–360. IEEE, 2017.

[25] A. Said, R. A. Abbasi, O. Maqbool, A. Daud, and N. R. Aljohani. Cc-ga: A clustering coefficient based genetic algorithm for detecting communities in social networks. *Applied Soft Computing*, 63:59–70, 2018.

[26] J. H. Seo and M. H. Kim. pm-scan: an i/o efficient structural clustering algorithm for large-scale graphs. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 2295–2298. ACM, 2017.

[27] H. Shiokawa, Y. Fujiwara, and M. Onizuka. Scan++: efficient algorithm for finding clusters, hubs and outliers on large-scale graphs. *Proceedings of the VLDB Endowment*, 8(11):1178–1189, 2015.

[28] T. R. Stovall, S. Kockara, and R. Avci. Gpuscan: Gpu-based parallel structural clustering algorithm for networks. *IEEE Transactions on Parallel and Distributed Systems*, 26(12):3381–3393, 2015.

[29] T. Takahashi, H. Shiokawa, and H. Kitagawa. Scan-xp: Parallel structural graph clustering algorithm on intel xeon phi coprocessors. In *Proceedings of the 2nd International Workshop on Network Data Analytics*, page 6. ACM, 2017.

[30] D. Wen, L. Qin, Y. Zhang, L. Chang, and X. Lin. Efficient structural graph clustering: an index-based approach. *Proceedings of the VLDB Endowment*, 11(3):243–255, 2017.

[31] S. White and P. Smyth. A spectral clustering approach to finding communities in graphs. In *Proceedings of the 2005 SIAM international conference on data mining*, pages 274–285. SIAM, 2005.

[32] X. Xu, N. Yuruk, Z. Feng, and T. A. Schweiger. Scan: a structural clustering algorithm for networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 824–833. ACM, 2007.

[33] Y. Xu, V. Olman, and D. Xu. Clustering gene expression data using a graph-theoretic approach: an application of minimum spanning trees. *Bioinformatics*, 18(4):536–545, 2002.

[34] H. Yin, A. R. Benson, J. Leskovec, and D. F. Gleich. Local higher-order graph clustering. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 555–564. ACM, 2017.

[35] K. R. Žalik and B. Žalik. Memetic algorithm using node entropy and partition entropy for community detection in networks. *Information Sciences*, 445:38–49, 2018.

[36] W. Zhao, G. Chen, and X. Xu. Anyscan: An efficient anytime framework with active learning for large-scale network clustering. In *2017 IEEE International Conference on Data Mining (ICDM)*, pages 665–674. IEEE, 2017.