A Novel Scalable Clustering Method for Distributed Networks

Wissm Inoubli, Sabeur Aridhi, Haithem Mezni, Mondher Maddouri, Engelbert Mephu Nguifo

A Novel Scalable Clustering Method for Distributed Networks

Wissm Inoubli[a,b], Sabeur Aridhi[b], Haithem Mezni[c], Mondher Madouri[d], Engelbert Mephu Nguifo[e]

[a] *University Tunis El-Manar, LIPAH, Tunis, Tunisia*
[b] *University of Lorraine, LORIA, France*
[c] *Higher Institute of Management, SMART Lab, Tunisia*
[d] *College Of Business, University of Jeddah, Saudi Arabia*
[e] *University Clermont Auvergne, CNRS, LIMOS, Clermont-Ferrand, France*

Abstract

Graph clustering is one of the key techniques to understand structures that are present in networks. In addition to clusters, bridges and outliers detection is also a critical task as it plays an important role in the analysis of networks. Recently, several graph clustering methods are developed and used in multiple application domains such as biological network analysis, recommendation systems and community detection. Most of these algorithms are based on the structural clustering algorithm. Yet, this kind of algorithm is based on the structural similarity, this later requires to parse all graph ' edges in order to compute the structural similarity. However, the height needs of similarity computing make this algorithm more adequate for small graphs, without significant support to deal with large-scale networks. In this paper, we propose a novel distributed graph clustering algorithm based on structural graph clustering. The experimental results show the efficiency in terms of running time of the proposed algorithm in large networks compared to existing structural graph clustering methods.

Keywords: Graph processing, Structural graph clustering, Big Graph Analysis, Community detection, Outliers detection, hubs detection

1. Introduction

Recently, the graph model has risen one of the most used data models in several applications like social networks (Said et al., 2018), road maps (Cao

and Krumm, 2009), pattern mining Fournier-Viger et al. and bioinformatics (Xu et al., 2002). For instance, a recent ranking shows that the popularity of graph databases model increased up to around 500% in the last years (Iyer et al., 2018). It has shown an optimum data model that allows to represent easily many relationships and facilitates 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 (Baborska-Narozny et al., 2016). This proliferation of a huge amount of data and the massiveness of graphs introduce additional factors to the renewed popularity of graph analytics (Iyer et al., 2018). Consequently, new applications and use cases have been mentioned in the literature. One important analysis technique in graph mining and graph analysis fields is graph clustering. Graph clustering helps identifying tightly connected regions within a graph (Žalik and Žalik, 2018; Günnemann et al., 2012). It has been used to solve various problems such as discovering communities in social networks and detecting protein complexes in Protein–Protein Interaction (PPI) networks (Xu et al., 2002).

It is important to mention that in some applications, the used graph could be very large. In such situation, the considered graph could be partitioned into several subgraphs and the computation is performed in a parallel/distributed way Dhifli et al. (2017); Aridhi et al. (2015). In this context, graph clustering algorithms are used to ensure the partitioning of the graph into several parts (Abbas et al., 2018; Yin et al., 2017).

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

- We propose a distributed graph clustering algorithm of large graphs (DSCAN). The proposed method is exact and allows discovering the same clusters that are discovered by SCAN, the reference algorithm for structural graph clustering (Xu et al., 2007).

- We conduct an extensive experimental study with large graphs, to evaluate the scalability of DSCAN. We compare DSCAN with four existing structural graph clustering algorithms.

The rest of the paper is organized as follows. After introducing graph clus-

tering in Section 1, we present a brief overview of related work in Section 2. In Section 3, we present the basic concepts related to the structural graph clustering. In Section 4, we present our proposed algorithms for large graph clustering. In Section 5, we provide the experimental results. The last section is devoted to the conclusion and the future work.

2. Related work

Several graph clustering algorithms have been proposed in the literature. Taking as examples, modularity-based approaches (LaSalle and Karypis, 2015) that represent an optimization solution of the modularity measure for each partitioning schema (generated randomly or according to a heuristic function) (LaSalle and Karypis, 2015). The Louvain method (Aynaud and Guillaume, 2010) represents one of the clustering algorithms based on modularity, that initially generates a random clustering. 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 (Kozawa et al., 2017). Graph partitioning (Brandes et al., 2003) and min-cut (Ding et al., 2001) are other methods used for the graph clustering which consist of splitting a graph into subgraphs while optimizing the cut edge during the partitioning. The spectral clustering (White and Smyth, 2005) is based on the graph density. It represents an input graph with a matrix and transforms this matrix so that to apply the basic clustering algorithm, like k-means (Hartigan and Wong, 1979).
A sampling-based distributed graph clustering method has been proposed in Sun and Zanetti (2019). The proposed algorithm is based on the density of the graph to produce a set of sparse subgraphs. Graph embedding has also been used in the graph clustering. In (Goyal and Ferrara, 2018), 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 behavior. To address this issue, the Structural Clustering Algorithm for Networks (SCAN) was proposed in (Xu et al., 2007) aiming, not only to identify the clusters in a 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 computation 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 literature, several works have been proposed for the structural graph clustering to overcome the drawback of SCAN. In (Shiokawa et al., 2015), Shiokawa et al. proposed an extension of the basic SCAN algorithm, namely SCAN++. The proposed algorithm aims to introduce a new data structure of directly two-hop-away reachable node set (DTAR). This new data structure is the set of two-hop-away nodes from a given node that are likely to be in the same cluster as the given node. SCAN++ could save many structural similarity operations, since it avoids several computations of structural similarity by vertices that are shared between the neighbors of a vertex and its two-hop-away vertices. In the same way, the authors in (Chang et al., 2016) 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, an algorithm named LinkSCAN* has been proposed in (Lim et al., 2014). 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, LinkSCAN* provides approximate results.

Other works have proposed parallel implementations of SCAN algorithm (Takahashi et al., 2017) (Mai et al., 2017) (Chang et al., 2017) (Stovall et al., 2015) (Wen et al., 2017). In (Takahashi et al., 2017), the authors proposed an approach based on openMP library (Bull, 1999). 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 (Mai et al., 2017) and (Chang et al., 2017), the authors have extended SCAN algorithm to deal with the addition or removal of edges. Authors in (Stovall et al., 2015), used 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.

On the other hand, an index approach is proposed in Wen et al. (2017) where the authors have emphasized the impact of the two parameters μ and ϵ on the running time of the structural graph clustering. Using the index approach a significant improvement has been shown b in terms of clustering running time. Again, the two sensitive parameters μ and ϵ have been taken into consideration by the authors in Wu et al. (2019). The main goal of the latter consists of estimate the best μ and ϵ values in order to enhance the accuracy and efficiency of the clustering results.

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/partitioned graphs. Through Table 1, we have summarized the discussed approaches according to some features.

As shown in Table. 1, 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 addition, these approaches are unable to process large graphs. Also in some applications, like social network, graphs are distributed by nature. Thereby, using the discussed algorithms, it should aggregate in one machine all partitions of a distributed graph in order to run the graph clustering. based on this limits, in this work, we tackle the problem of large and dynamic graph clustering in a distributed setting.

3. Background

3.1. Structural graph clustering: Basic concepts

Graph clustering consists in dividing a graph into several partitions or subgraphs. 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 that are 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

Table 1: Comparative study on existing graph clustering methods

Approach	**Parallel**	**Distributed**	**Processing model**	**Graph partitioning**	**Main contributions**
SCAN (Xu et al., 2007)	No	No	Sequential	No	Basic implementation of structural graph clustering
SCAN++ (Shiokawa et al., 2015)	No	No	Sequential	No	Reducing the number of similarity computations
AnySCAN (Zhao et al., 2017)	Yes	No	Parallel	No	Parallelizing SCAN
pSCAN (Chang et al., 2016)	Yes	No	Parallel	No	Reducing the number of similarity computations
Index-based SCAN (Wen et al., 2017)	No	No	Sequential	No	Interactive SCAN
ppSCAN (Che et al., 2018)	Yes	No	Parallel	No	Parallel version of pSCAN
SCAN based on GPU (Stovall et al., 2015)	Yes	No	Parallel	No	A GPU-based version of SCAN
pm-SCAN (Seo and Kim, 2017)	Yes	No	Parallel	Yes	Graph partitioning to reduce the I/O costs
dpSCAN (Wu et al., 2019)	No	No	Sequential	No	Clustering results without μ and ϵ parameters

edges between vertices. Each of those elements can represent a real property in real-word applications. Let u and v be two vertices in V. We denote by (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,is denoted by $N(v)$, and represents 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 (denoted by ϵ-neighborhood) between them. SCAN uses a ϵ threshold and Eq. (3) to filter, for each vertex, its strongest connections. The ϵ-neighborhood is defined as follows.*

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

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 which 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 is modeled as follows:

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

Definition 3.5. *(Border) Let v_c be 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 main steps of clusters' building algorithm are the following: 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, the algorithm checks if the list of borders has a core vertex. Then, it inserts their borders into the same cluster and it applies this step recursively until adding all the borders of the connected cores. Finally, the algorithm chooses other core vertices and applies the same previous steps until checking all core vertices.*

Among the fundamental information returned by SCAN, compared to other graph clustering algorithms, we mention bridge and outlier information, 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 join any cluster. In this context, SCAN algorithm classifies these vertices into 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 bridge. Otherwise it is considered as an outlier.

Algorithm 1: Basic SCAN algorithm

Input : A Graph G and parameters (μ ,ϵ)
Output: $\mathbb{C}, \mathbb{B}, \mathbb{O}$

```
foreach (u, v) ∈ E do
    Compute σ(u, v)
end
Core ← ∅
foreach u ∈ V do
    if (|N_ε(u)| ≥ μ) then
        Core = Core ∪ {u}
    end
end
Cluster ← ∅
foreach unprocessed core vertex u ∈ Core do
    Cluster ← {u}
    Mark u as processed
    foreach unprocessed border of vertex v ∈N_ε(u) do
        Cluster←Cluster ∪ {v}
        Mark v as processed
    end
end
```

3.2. 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, lines 1-3 of Algorithm 18 use Eq. (2) to compute the structural similarity for each edge $e \in \mathbb{E}$. Then, Eq. (3) (lines 5-8) is used to define the core vertices (see gray vertices in Fig. 1). After that, we proceed to the clustering step, then we apply Definition 3.6 (lines 11-16) to build the clustering schema. In our example we have four core vertices: 0,2,9 and 1O. Each core has a list of border vertices (the neighbors that have strong connections with a core). In our example, vertex number 2 is a core. This later has the vertices number 1, 4, 5, 3 and 0 as the list of borders since they have strong connections with the core. The core and its borders build a cluster, and if a border is a core we join all its borders into the cluster. Like in our example, the vertex 2 is a core. Hence, we join 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 vertex 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, whereas 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.

4. 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 (Aridhi et al., 2017). This latter is 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 (subgraphs in our case). Each chunck/partition is assigned to a worker, which performs a specific computation. The master machine orchestrates the workers' execution. In the

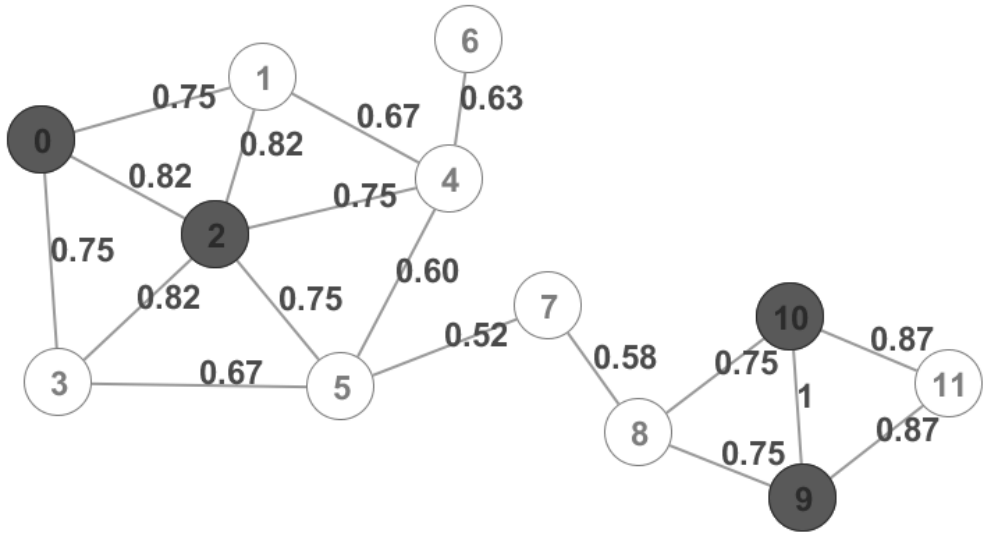

Figure 1: Running example ($\epsilon = 0.7$ and $\mu = 3$).

following, we present the main two steps of DSCAN: (1) distributed graph partitioning and (2) distributed graph clustering.

4.1. Distributed graph partitioning

In this step, we split the input graph G into several small partitions $P_1, P_2, ..P_n$, 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 (Dhillon, 2001). 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.

Algorithm 2: Distributed graph partitioning

Input : Graph file GF as a text file, parameter (NP number of partitions)

Output: $\mathbb{P}$ set of partitions

1 *BLADYG initialization according to the NP parameter*
2 *Master machine: split GF into a set of sub-files* $\mathbb{GF}$
3 **foreach** $GF_i \in \mathbb{GF}$ **do**
4 | *Assign* GF_i *to worker* W_i
5 **end**
`/* In parallel */`
6 **foreach** $Worker\ W_i \in \mathbb{W}$ **do**
7 | *Get list of vertices and edges from* GF_i
8 | *Find the list of frontier vertices from the neighbor workers*
9 **end**
`/* In parallel */`
10 **foreach** $Worker\ W_i \in \mathbb{W}$ **do**
11 | **foreach** $Edge \in \mathbb{E}$ **do**
12 | | *(a,b)=Edge ;*
13 | | *Let* P_i^f *the frontier vertices*
14 | | **if** *(*$a \in P_i^f \cup b \in P_i^f$ *)* **then**
15 | | **end**
16 | | *Set E as a cut edge*
17 | **end**
18 **end**

4.2. Distributed graph clustering

DSCAN has two main steps: (1) local clustering step and (2) merging step.

Step 1: Local clustering. As presented in Algorithm 3, the input graph G is splitted into multiple subgraphs/partitions ($\mathbb{P}$), each one is assigned to a worker machine. The partitioning step, as mentioned in Section 4.1,

Algorithm 3: DSCAN

Input : Graph G , parameters (μ, ϵ, α)
Output: $\mathbb{C}lusters, \mathbb{B}ridges, \mathbb{O}utliers$

`/* Divide G into subgraphs` $\mathbb{G} = \{G_1 G_2 .. G_\alpha\}$ `according to parameter` α `*/`
$\mathbb{P} \longleftarrow$ ***Partition*** *(G,α)*
`/* In parallel */`
foreach $P_i \in \mathbb{P}$ **do**
  Assign P_i to W_i
end
`/* In parallel */`
`/* Step 1: Local clustering */`
foreach *Worker* $W_i \in \mathbb{W}$ **do**
  Let P_i the current partition
  Find the frontier vertices in P_i and duplicate them into neighbor 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 exchange theirs local clusters between them; using Worker2Worker message
foreach *Worker* $W_i \in \mathbb{W}$ **do**
  if *($C_1 \cap C_2 \cap .. \cap C_\alpha = \mathbb{V}$; and $\exists V_i \in \mathbb{C}ore$* **then**
    $C \leftarrow$ ***Merge***$(C_1, C_2, ..C_\alpha)$
    Send C to the master
  end
  else
    Send local clusters to master
  end
  for V_i ***IN*** $\mathbb{O}utliers$ **do**
    if *($V_i \in \mathbb{C}ore \cup \mathbb{B}order \cup \mathbb{B}ridges$)* **then**
      Remove *V_i from the list of $\mathbb{O}utliers$*
    end
  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
    end
    if *($Nb_{Connections} \geq 2$)* **then**
      Add *V_i to $\mathbb{B}ridges$*
      Remove *V_i from $\mathbb{O}utliers$*
    end
  end
end
Send Clusers,Bridges,Outliers to the master using Worker2Master message

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 the graph consistency will be ensured during the partitioning step.

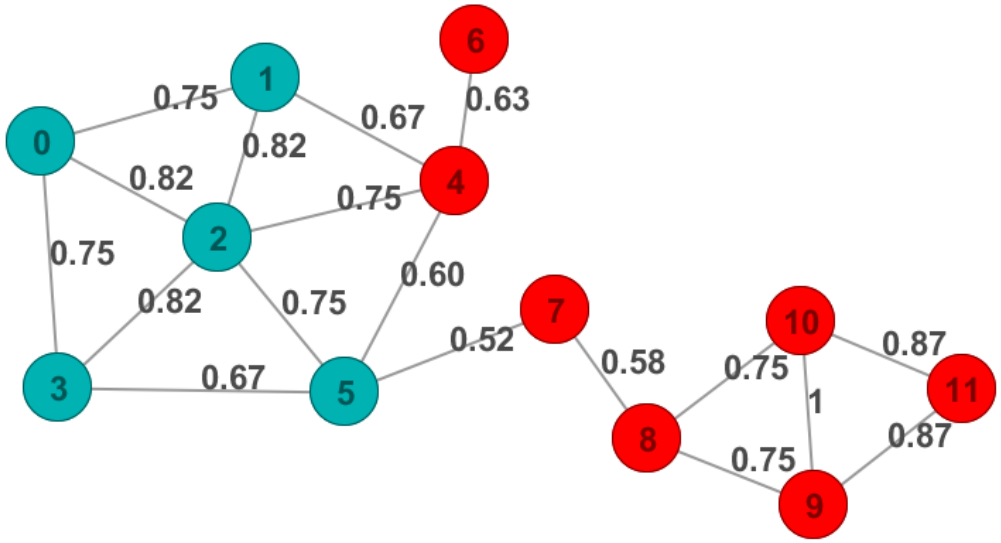

Figure 2: Illustrative example

Assume that a graph G is partitioned into two 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 they are denoted by V_P^f. For example, V_{P1}^f={1,2,5} and V_{P2}^f={4,7}. Each $v \in V_P^f$ has a set of internal neighbors and external neighbors. For example, vertex 4 is a vertex of the partition $P2$. It has the vertex 6 as internal and the vertices 1,2 and 5 as external neighbors. Computing the structural similarity of a frontier vertex u considers that all the neighbors of u belong to the same partition. Thereby, we duplicate all frontier vertices in partition P to all neighbor partitions and we call them external vertices. For example in our running example, $P1$ has frontier vertices V_{P1}^f={1,2,5} 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 $P1$ and $P2$, we will find several conflicts such as the vertex $v \in V_{P1}^f$ is a core vertex in $P1$, and an outlier in $P2$. 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 defined a set of scenarios regarding the merging step. These scenarios will repetitively be applied to every two partitions of G, until combining all the partitions (see Algorithm 3, lines 16-40). 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 achieves several scenarios (Lemmas 4.1, 4.2 and 4.3) to solve the 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 into the same cluster.

Lemma 4.2. ***(Outlier to Bridge)*** *Let $\mathbb{C}_1$ and $\mathbb{C}_2$ two sets of local clusters in partitions P_1 and P_2, respectively. $\exists$ C_1 and C_2 two clusters that 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, with $N(o) \cap c_1 \neq \emptyset$ 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 weakly connected with the two clusters C_i and C_j. Hence, according to the Definition 3.7, o should be changed to a bridge vertex.

Lemma 4.3. ***(Bridge to Outlier)*** *Let c_1 and c_2 two local clusters in the partitions P_1 and P_2, respectively.*
$\exists$ a bridge b according to only two clusters c_1 and c_2, when c_1 and c_2 two

clusters that will be merged into one cluster, 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 a set of bridges with other local clusters, and $\exists\, b$ a bridge vertex according to the two clusters C_i and C_j only, 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, then they will be merged into a single cluster. In this case, $(C_i,C_j) \Rightarrow C$ which makes b be weakly connected with only one cluster C, then according to Definition 3.7, b should change its status from bridge to outilier.

For instance looking at 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 one single cluster. Subsequently, in lines 23-27, 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 the 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.

(a) Partition 1

(b) Partition 2

Figure 3: The partitioned graph G used in the running example in Section 3

4.2.1. Illustrative example

Fig. 3 shows a demonstrative example of a graph clustering using DSCAN. In this example, we use the same parameters (ϵ and μ) of the running example in Section 3, and two partitions ($P1$ and $P2$). 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, the blue vertices 1,2 and 5 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 the workers perform the similarity computation, check the status for each vertex and build the local clusters, as it is 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 $P2$. As for $P2$, it 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. This latter is a bridge according to basic SCAN (see running example in Section 3). 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.

4.2.2. Time complexity analysis of DSCAN

Let $\mathbb{E}$ be the set of all edges (internal and cut edges), $\mathbb{V}$ be the set of all vertices(internal and external) and the set of core vertices is $\mathbb{V}_c$. The time complexity of DSCAN is like basic SCAN, and can be divided into three step, (i) the structural neighborhood step, when DSCAN according to Definition 3.1 aggregates for each vertex its list of neighbors. Thus, the complexity is of the order of $\mathcal{O}((v_i, v_j)_{i,j=1}^{|V|}, (u_i, u_j) \in \mathbb{E})$=$\mathcal{O}(|\mathbb{E}|)$. (ii) The time complexity of the computation of structural similarity which is defined by Definition 3.2. In this step, DSCAN enumerates for each edge (u, v) the set of common neighbors. Therefore, the time complexity is $\mathcal{O} = (min(|N(u)|, |N(v)|).|\mathbb{E}|)$. (iii) DSCAN perfoms the clustering step using the μ in order filter the core vertices, and groups those which have strong connections into a same cluster. Like this, the complexity is $\mathcal{O} = (|V_c|-|(v_i, v_j)|_{i,j=1}^{|V|}, v_i, v_j \in V_c$ and $\sigma(v_i, vj) \geq \epsilon)$.

5. Experiments

In this section, we present our experimental study that evaluates the effectiveness and efficiency of our proposed algorithm for structural clustering of large and distributed graphs. In this experiment, we performed some experiments like the impact of graph size and the number of machines in the cluster, We also evaluated some features related to DSCAN.

5.1. Experimental protocol

In the first experiment part, 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].
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 pp-SCAN. 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 16.04 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. `https://discan.yo.fr/DGC.html` provides some details about the configuration, deployment of DSCAN, and also a user guide is presented.

5.2. Experimental data

For all test cases of static graphs, we used real-world graph datasets (see Table 2) obtained from the Stanford Network Analysis Project (SNAP) (Leskovec and Krevl, 2015).

5.3. BLADYG framework

BLADYG is a distributed and parallel graph processing framework that runs on a commodity hardware. The architecture of BLADYG is based on a master/slaves topology. BLADYG starts by reading the input graph from many different sources, which can be local or distributed files such as

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

Table 2: 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

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 useful data during the graph 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 own partitioning techniques.

5.4. Experimental results

Speedup. We evaluated the speedup of DSCAN compared to the basic SCAN and its variants presented in Section 5.1. The compared algorithms use different graph representations. In fact, AnyScan and SCAN implementations use the adjacency list representation (Doerr and Johannsen, 2007), whereas both pSCAN and ppSCAN use the Compressed Sparse Row (CSR) format (D'Azevedo et al., 2005). In our proposed algorithm, we used an edge list format, in which each line represents one edge of the graph. 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 equipped with an 8 vCPU and 32 GB of main memory.

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

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 especially with pSCAN and ppSCAN. On the other hand, this gap become reduced when the graph size increases (case of G4 dataset). The plots bars in Fig. 4 shows a gap of 12x between DSCAN

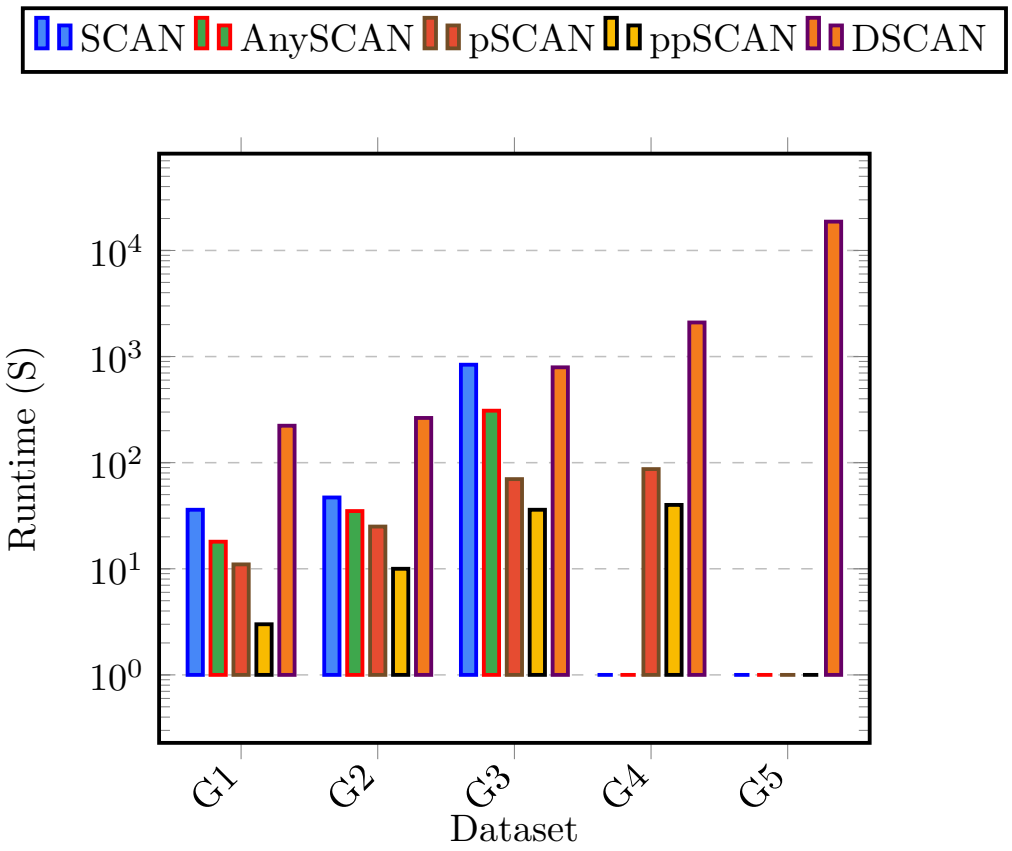

Figure 4: Impact of the graph size on the processing time of both SCAN and DSCAN

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 reaching 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. Fig 4 also 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 hori-

zontal 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 for 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 of about 82% in the response time, 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. This is the case of California road network which is smaller than LiveJournal graph. The red curve's behavior can be explained by the

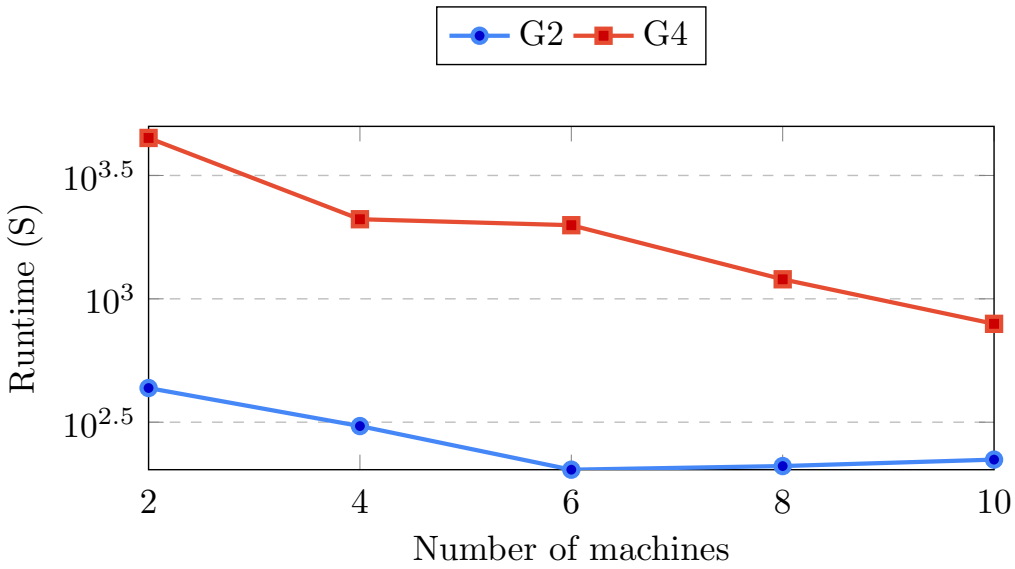

Figure 5: Impact of the number of workers on the running time (with $\epsilon = 0.5$, $\mu = 3$).

splitting of the input graph into small sub-graphs and by performing a local clustering on each subgraph, which reduces the global response time even with the additional cost of aggregating the intermediate results returned by each machine in the cluster. That was not the case with the 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 noted 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 numbers of clusters, bridges and noise vertices depend on the values of ϵ and μ. In ppSCAN, when we decrease 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 for different graph data sets. As shown in Fig. 6, the response time is slightly

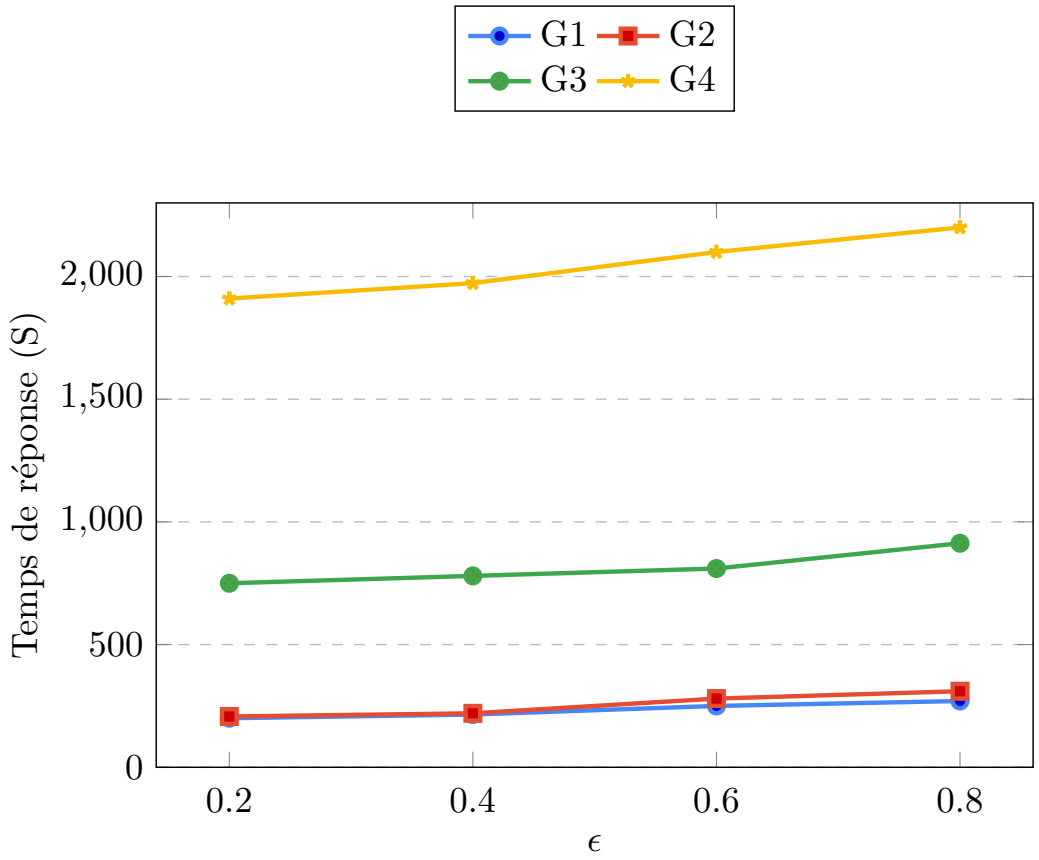

Figure 6: Impact of ϵ size on the running time of DSCAN

dependent on the value of ϵ, especially with large graphs (G3 and G4). 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 we did not see the impact of ϵ on the previous steps (graph loading and

local clustering). 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. In fact, when the value of ϵ increases, the number of outliers becomes larger. Also, in the merging function (see Algorithm 3, lines 17-40), DSCAN combines the local results by starting to check the clusters that share almost one core, in order 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 in DSCAN.

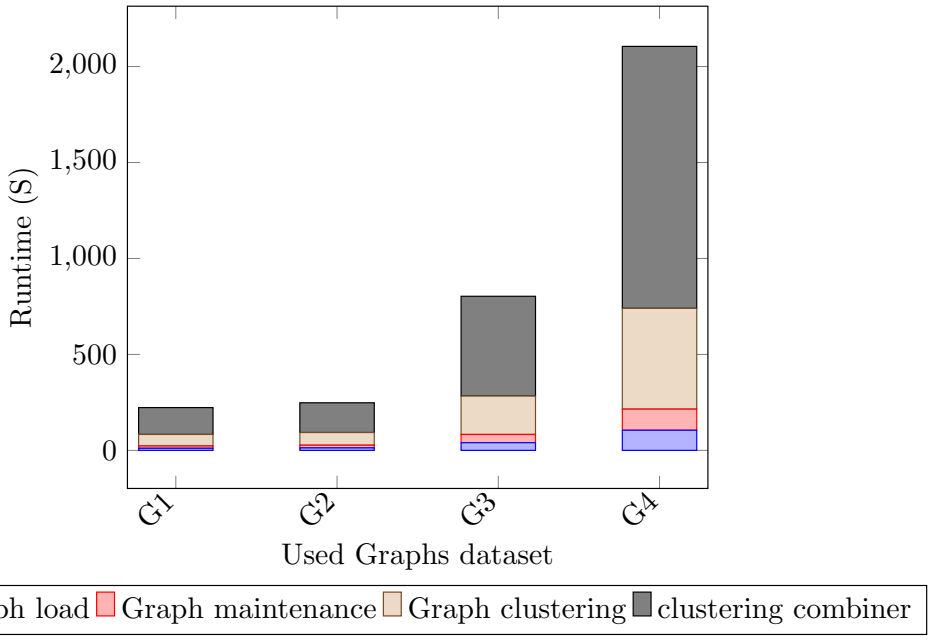

Figure 7: Performance of DSCAN phases

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 load-

ing step and the duplication of vertices in frontiers to ensure the consistency property, as we discussed in Section 4.1. 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, ϵ=0.5 and 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 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.

6. Conclusion

In this paper, we proposed DSCAN, a distributed algorithm for big graph clustering based on the structural similarity. DSCAN is build on top of 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 deal with any graph size and is scalable with a large number of machines, in a parallel way. We have presented the main functions of DSCAN starting from the partitioning to the combining of intermediate results for each worker. Also, we have performed an extensive experimentation about our proposed algorithm, compared with other ones. The experiments have shown that DSCAN featured an horizontal scalability that is not guaranteed with other algorithms.

In our future works, we will 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

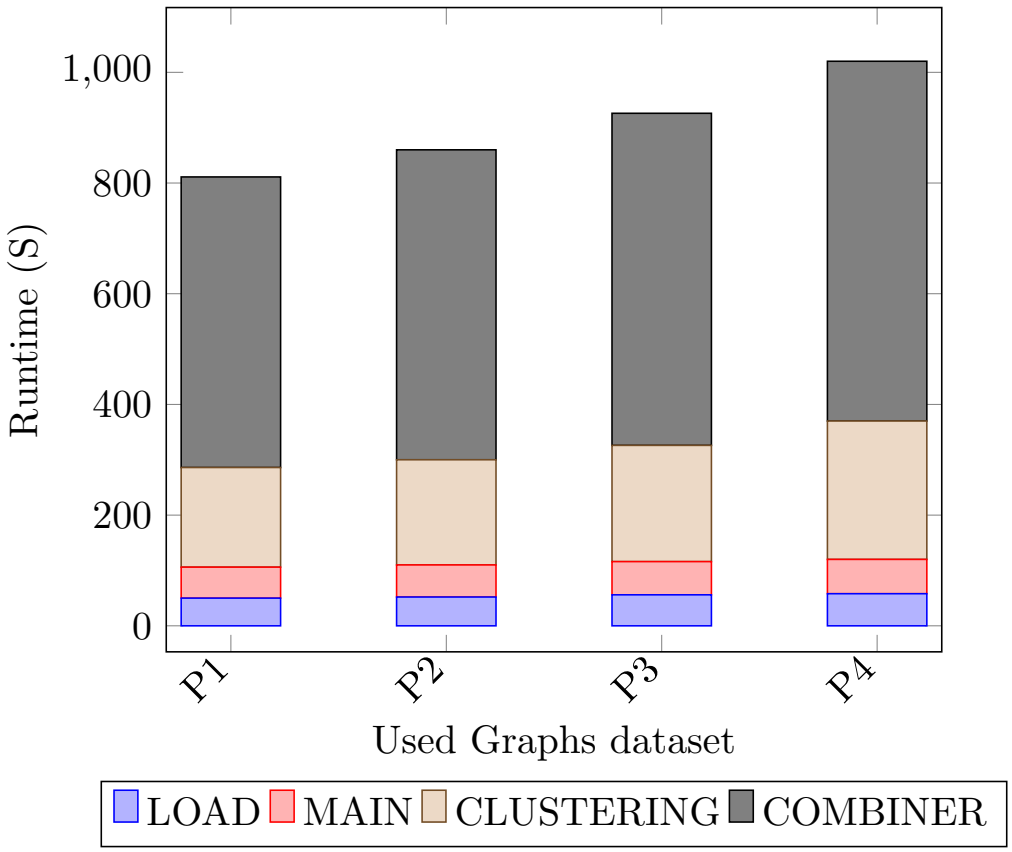

Figure 8: Impact of the partitionning step on DSCAN response time (ϵ=0.5, μ=3)

partitioning, whereas for the dynamic graph clustering, we plan to make DSCAN support big evolving graphs in order to check all the snapshots of a graph. Having this hand, we can evaluate and follow the evolution of each cluster and other vertex types (border, outlier, and bridge).

References

Abbas, Z., Kalavri, V., Carbone, P., Vlassov, V., 2018. Streaming graph partitioning: an experimental study. Proceedings of the VLDB Endowment 11, 1590–1603.

Aridhi, S., d'Orazio, L., Maddouri, M., Nguifo, E.M., 2015. Density-based

data partitioning strategy to approximate large-scale subgraph mining. Information Systems 48, 213–223.

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

Aynaud, T., Guillaume, J.L., 2010. Static community detection algorithms for evolving networks, in: 8th International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks, IEEE. pp. 513–519.

Baborska-Narozny, M., Stirling, E., Stevenson, F., 2016. 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, ACM. p. 17.

Brandes, U., Gaertler, M., Wagner, D., 2003. Experiments on graph clustering algorithms, in: European Symposium on Algorithms, Springer. pp. 568–579.

Bull, J.M., 1999. Measuring synchronisation and scheduling overheads in openmp, in: Proceedings of First European Workshop on OpenMP, Citeseer. p. 49.

Cao, L., Krumm, J., 2009. From gps traces to a routable road map, in: Proceedings of the 17th ACM international conference on advances in geographic information systems, ACM. pp. 3–12.

Chang, L., Li, W., Lin, X., Qin, L., Zhang, W., 2016. pscan: Fast and exact structural graph clustering, in: 2016 IEEE 32nd International Conference on Data Engineering (ICDE), IEEE. pp. 253–264.

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

Che, Y., Sun, S., Luo, Q., 2018. 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, pp. 77:1–77:10. URL: http://doi.acm.org/10.1145/3225058.3225063, doi:10.1145/3225058.3225063.

Dhifli, W., Aridhi, S., Nguifo, E.M., 2017. Mr-simlab: Scalable subgraph selection with label similarity for big data. Information Systems 69, 155–163.

Dhillon, I.S., 2001. 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, ACM. pp. 269–274.

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

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

D'Azevedo, E.F., Fahey, M.R., Mills, R.T., 2005. Vectorized sparse matrix multiply for compressed row storage format, in: International Conference on Computational Science, Springer. pp. 99–106.

Fournier-Viger, P., He, G., Cheng, C., Li, J., Zhou, M., Lin, J.C.W., Yun, U., . A survey of pattern mining in dynamic graphs. WIREs Data Mining and Knowledge Discovery n/a, e1372.

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

Günnemann, S., Boden, B., Seidl, T., 2012. Finding density-based subspace clusters in graphs with feature vectors. Data mining and knowledge discovery 25, 243–269.

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

Iyer, A.P., Panda, A., Venkataraman, S., Chowdhury, M., Akella, A., Shenker, S., Stoica, I., 2018. 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), ACM. p. 10.

Kozawa, Y., Amagasa, T., Kitagawa, H., 2017. Gpu-accelerated graph clustering via parallel label propagation, in: Proceedings of the 2017 ACM on Conference on Information and Knowledge Management, ACM. pp. 567–576.

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

Leskovec, J., Krevl, A., 2015. {SNAP Datasets}:{Stanford} large network dataset collection .

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

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

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

Seo, J.H., Kim, M.H., 2017. 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, ACM. pp. 2295–2298.

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

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

Sun, H., Zanetti, L., 2019. Distributed graph clustering and sparsification. ACM Trans. on Parallel Computing (TOPC) 6, 1–23.

Takahashi, T., Shiokawa, H., Kitagawa, H., 2017. Scan-xp: Parallel structural graph clustering algorithm on intel xeon phi coprocessors, in: Proceedings of the 2nd International Workshop on Network Data Analytics, ACM. p. 6.

Wen, D., Qin, L., Zhang, Y., Chang, L., Lin, X., 2017. Efficient structural graph clustering: an index-based approach. Proceedings of the VLDB Endowment 11, 243–255.

White, S., Smyth, P., 2005. A spectral clustering approach to finding communities in graphs, in: Proceedings of the 2005 SIAM international conference on data mining, SIAM. pp. 274–285.

Wu, C., Gu, Y., Yu, G., 2019. Dpscan: Structural graph clustering based on density peaks, in: International Conference on Database Systems for Advanced Applications, Springer. pp. 626–641.

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

Xu, Y., Olman, V., Xu, D., 2002. Clustering gene expression data using a graph-theoretic approach: an application of minimum spanning trees. Bioinformatics 18, 536–545.

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

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

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