Mathematics and Computer Science
2022; 7(1): 1-8
http://www.sciencepublishinggroup.com/j/mcs
doi: 10.11648/j.mcs.20220701.11
ISSN: 2575-6036 (Print); ISSN: 2575-6028 (Online)

An Overview of Metrics for Evaluating the Quality of Graph Partitioners

Mustapha Abdulkadir Sani[1, *], Abdulmalik Ahmad Lawan[1], Ayaz Khalid Mohammed[2], Abdulkadir Ahmad[1], Yusuf Haruna[1]

[1]Department of Computer Science, Kano University of Science and Technology, Wudil, Nigeria

[2]Department of Computer Science, University of Zakho, Kurdistan, Iraq

Email address:

mustapha.abdulkadir@kustwudil.edu.ng (M. A. Sani), aalawan@kustwudil.edu.ng (A. A. Lawan), ayaz.bilir@uoz.edu.krd (A. K. Mohammed), abdulkadir.ahmad@kustwudil.edu.ng (A. Ahmad), yusuf.haruna@kustwudil.edu.ng (Y. Haruna)

[*]Corresponding author

To cite this article:

Mustapha Abdulkadir Sani, Abdulmalik Ahmad Lawan, Ayaz Khalid Mohammed, Abdulkadir Ahmad, Yusuf Haruna. An Overview of Metrics for Evaluating the Quality of Graph Partitioners. *Mathematics and Computer Science.* Vol. 7, No. 1, 2022, pp. 1-8.
doi: 10.11648/j.mcs.20220701.11

Received: January 12, 2022; **Accepted**: February 3, 2022; **Published**: February 16, 2022

Abstract: Nowadays, many applications that involve big data can be modelled as graphs. In many cases these graphs may be too large to be loaded and processed on a single commodity computer. This necessitated the development of frameworks that allow processing large graphs by distributing the graph among nodes in a cluster. To process a graph using these frameworks, first, the graph is partitioned into smaller components called subgraphs or partitions, and then assign these smaller subgraphs to different nodes for parallel processing. Depending on the type of processing (example, computing pagerank, counting number of triangles etc.), there will be some communication between nodes during the execution, this communication affects execution time. Therefore, graph partitioning is an important step in distributed graph processing. Being able to determine the quality of a partition prior to processing is important as this will allow us to predict the execution time before the actual processing. A number of metrics for evaluating the quality of a graph partitions exist, but studies show that these metrics may not serve as accurate predictors in many cases. In this work, we reviewed published papers about graph partitioning and we were able to identify and defined more metrics in order to have a catalogue of these metrics.

Keywords: Graph, Graph Partitioning, Partitioning Algorithms, Metrics

1. Introduction

Nowadays big data is represented as graphs which is often too large to load and process on a single machine. Examples of applications that are modelled as graphs include online social networks (like Facebook, Twitter, Instagram, LinkedIn, MySpace, etc), and web graphs. In the case of online social networks vertices are used to represent users while edges are used to represent the relationship between users, while in the case of web graphs web pages are represented as vertices, and links between them are represented the edges of the graph. The most well-known web graph is www, which contains more than 50 billion vertices (web pages) and more than 1 trillion edges (links) [1].

Most of popular online social networks consist of millions of vertices and billions of edges. As at December, 2014 Facebook has 1.39 billion active users (vertices) with more than 400 billion relations (edges). Likewise in case of twitter, as at March, 2015 twitter has 288 million active users (vertices) with an estimated total of 60 billion followers (edges) [2]. These graphs are normally stored in a text file before performing computation on the graph. In many cases, the size of a graph file may be in hundreds of gigabytes, making it almost impossible to load and process the graph using a single computer [3].

Although some frameworks that enable processing large graph using single computer exists, but the traditional way to handle large graph is to use a cluster of nodes. First, the graph is partitioned into smaller components called partitions or subgraphs, and then assign each partition is assigned to a node in the cluster. Each node will process a subset of the

graph in parallel [4]. Unlike normal parallel data processing where a large database can be partitioned and each partition is processed independent of each other, in parallel processing of partitioned graph, the nodes need to periodically share intermediate computation result during execution. This sharing of computation results introduces communication overhead which has a significant impact on the execution time [4]. By partitioning the graph in an efficient manner, we will be able to mitigate this communication overhead, which will also lead to reduction of total execution time. In view of this it is essential to maintain the locality of information while distributing the graph among the multiple nodes by trying to have a minimum number of vertices or edges that are cut during partitioning. It is also important to partition the graph into equal subsets so as to distribute the computational load evenly among nodes [5]. In practice partitioning a graph into subgraphs is a hard problem, existing Partitioning Algorithms rely on heuristics to find to find good enough tradeoffs between balance (splitting the graph into equal subsets) and communication (minimizing number of edges or vertices that are cut) [4].

1.1. Organization of the Paper

The paper is organized as follows; Section one gives the general introduction, in section two we discussed two classical techniques of graph partitioning (i.e. *edge-cut* partitioning and *vertex-cut* partitioning). The problem statement of graph partitioning is presented in section three. We presented the motivation of the study in section four. Overview of identified metrics is presented in section five, and finally, we draw conclusion in section six.

1.2. Our Contribution

Our major contribution is that we were able to compile and document a considerable number of evaluation metrics which are used by so many and different researchers in the area of graph processing to evaluate the quality of graph partitions produced by their graph partition algorithms. Our work is going to be useful to researchers who want to propose a new graph partition algorithm or those researchers that want to use existing one. To the best of our knowledge, we are the first to conduct this type of study.

2. Techniques of Graph Partitioning

There are two major techniques for partitioning a graph into subgraph depending on weather the subgraphs are composed of disjoint set of vertices or disjoint set of edges. The former is known as *edge-cut* partitioning and the latter is known as *vertex-cut* partitioning.

2.1. Edge-cut Partitioning

This technique also called *vertex partitioning* [6], is the classical way of partitioning graphs [4, 5]. In this approach, vertices of a graph are divided into disjoint sets of nearly the same size, while minimizing number of edges that span over separated sets. An edge is cut if its vertices are assigned to two different partitions [4].

While a good edge cut partitioning can reduce communication overhead and can also balance the number of vertices in each subgraph, some studies [4, 5, 7] show that it does not produce good partitions on real world graphs which normally follow power-law degree distribution.

2.2. Vertex-cut Partitioning

It was proved in Gonzalez [7] that real-world graphs like online social networks can be partitioned efficiently if *vertex cut* approach also known as *edge partitioning* [6] is used. In *vertex cut* approach, edges of a graph are split into different disjoint sets of nearly the same size while minimizing number of vertices that are cut. In this approach vertices may be cut and thus replicated in more than one partition due to the distribution of their edges across different partitions. A good *vertex cut* is the one that achieve minimum replica while maintaining the balanced number of edges among partitions. Some graph processing frameworks like GraphX, Power Graph are based on *vertex-cut* technique [4, 6, 8, 9]. Figures 1(a) and 1(b) corresponds to *edge-cut* and *vertex-cut* partitioning respectively.

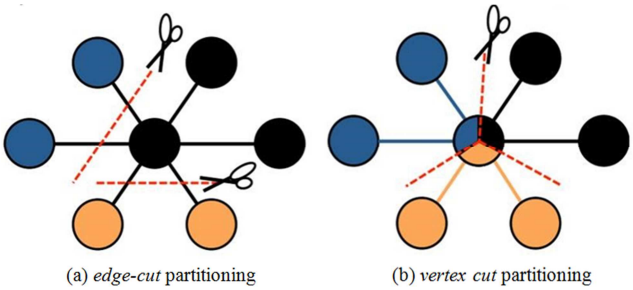

Figure 1. *Partitioning a graph into three partitions [10].*

3. Problem Statement

As we have mentioned in section 2 that there are to major techniques of partitioning a graph which are *vertex-cut* and *edge-cut* portioning, in this section we give the problem statement for the two techniques.

3.1. Problem Statement for Edge-cut Partitioning

In the case of *vertex-cut* partitioning, the graph partitioning problem is defined as:

Given an input graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, a *vertexcut* partitioning divides the set of edges E into N disjoint subsets of nearly equal size, $E_1, E_2, \ldots E_N$. Each partition also has a subset of vertices that hold at least one of the edges in that partition. This means that a vertex is part of a partition E_i if it is either source or destination of an edge in E_i. This implies that an edge can only be in one partition but a vertex can at least belong to one partition or at most to N partitions. This problem targets minimizing the number of vertices that are cut while maintain balance [4].

3.2. Problem Statement for Edge-cut Partitioning

Similarly, in the case of *edge-cut* partitioning the partitioning problem is defined as:

Given an input graph $G = (V, E)$, where V is the set of vertices and E is the set of edges, an *edgecut* partitioning divides the set of vertices into N disjoint subsets of nearly equal size $V_1, V_2, \ldots V_N$. Using this partitioning a vertex can only belong to one partition but edges can cross the boundaries of partitions. The problem always targets minimizing the number of edges that cross boundaries while maintaining balanced number of vertices among partitions [5].

3.3. Metrics for Evaluating the Quality of Partition

Several metrics for evaluating the quality of partitions produced by partitioning algorithms were considered in Mykhailenko [4] and Mykhailenko [6]. These metrics are categorized into two categories namely, *execution metrics* and *partition metrics* [6].

3.3.1. Execution Metrics

Execution metrics are those metrics that can only be used after processing the graph. Examples of execution metrics include partitioning time, processing time, number of rounds performed by the partitioner, and network communication overhead. According to Mykhailenko [6], a limitation of execution metrics is that they are tightly coupled to specific applications and execution environments, making them not suitable for general comparisons. Another limitation according to them is that they can also be costly to evaluate and measure.

3.3.2. Partition Metrics

Due to limitations of execution metrics mentioned above, another category of metrics known as *partition metrics* has been proposed [4], [6]. *Partition metrics* enable us to evaluate the quality of partitions produced prior to actual graph processing. *Partition metrics* are suitable for general comparison and are less costly to evaluate and measure. More importantly *partition metrics* can be used to predict execution time in some cases, although findings from Mykhailenko [4] shows that, known partition metrics may not serve as good predictors of execution time in some other cases, we are going to elaborate in this section. In this work we are going to focus on *partition metrics*.

Mykhailenko [4] defined and studied six (6) partition metrics. The main objective of their study was to investigate and find out if it is possible to predict execution time prior to actual graph processing by evaluating the quality of a partitions produced using the *partition metrics*. According to them, this answer will be useful for a data analyst who wants to choose a particular partitioner suitable for his/her graph and for a developer who wants to proposed more efficient partitioner. To answer this question, they experimented with Apache GraphX, and performed accurate statistical analysis. The following are the *partition metrics* they considered.

Balance which they denoted as *BAL* was used to measure the ratio between maximum number of edges in one partition and average number of edges in all subsets. It is defined as:

$$Bal = \frac{max_{i=1\ldots N}\,(|E_i|)}{|E|/N}$$

Normalized standard deviation denoted as *NSD* shows the standard deviation of a partition size. They defined it as:

$$NSD = \sqrt{\sum\left(\frac{|Ei|}{E/N} - 1\right)^2 \frac{1}{N}}$$

Largest Partition which was denoted by *LP* was used to show the number of vertices in the largest subset. It is defined as:

$$LP = max_{\,i=1\ldots N}\,|(E_i)|$$

Vertex Cut which was denoted as *VC* was used to show the number of verities that were cut during the partitioning. It is defined as:

$$VC = |V| - \sum_{i=1}^{N} |\overline{F}(E_i)|$$

Replication Factor denoted as *RF* shows the ratio between the total number of vertices in the portioned graph and the main graph.

$$RF = \sum_{i=1}^{N} |V(E_i)| \frac{1}{|V|}$$

Communication Cost denoted as *CC* represented the total number of frontier vertices among all the subsets.

$$CC = \sum_{i=1}^{N} |F(E_i)|$$

Mykhailenko [4] refer to the vertices that appear in more than one partition as frontier vertices. Each frontier vertex is cut at least once.

Mykhailenko [4] further divided these partition metrics into two categories with each category containing three set of metrics. The first three metrics targeted on quantifying how uniform are the sizes of partitions and how processing can be balanced across the nodes in the cluster. The second category, which contains the last three metrics, targeted the communication overhead by considering the vertices that are replicated in different partitions. According to them Apache GraphX does not have built-in functions which compute these metrics, that is why they have implemented new GraphX functions to compute them.

4. Motivation of the Study

The findings in Mykhailenko [4] show that there is still need to investigate and find more metrics, since the metrics they considered do not capture all properties of graph. These six metrics may not serve as accurate predictors of execution time in some cases, especially if the graph we are processing has some features that are not captured by these metrics. The main motivation of this research is to study large volume of published papers on graph partitioning with a view to define and analyze

more partition metrics which may likely capture the features of a graph that were not captured by the known metrics.

To accomplished the task so many papers about graph partitioning were studied, but we discovered that there are a very few research work dedicated to *vertex-cut* partitioning. Majority of the published work focus on *edge-cut* partitioning.

In this section we present what we found from the papers studied. Among all the papers, we studied that only two are based on *vertex cut* partitioning.

5. Overview of Identified Metrics

In this section, we give a detailed overview of the metrics we identified in the course of the research. A lot of papers were studied, we categorized the papers into two categories, those that focus on *vertex-cut* partitioning and those that focus on *edge-cut* partitioning.

5.1. Vertex Cut Metrics

In this sub-section we present our findings from papers that talked about *vertex-cut* partitioning.

Kim [11] proposed partitioning algorithm based on vertex-cut. The main aim of their algorithm was to ensure that the resulting partitions are well balanced (i.e of the same size) and at the same time cutting minimum number of vertices. Their algorithm was able to find a set of vertices which are used to efficiently and effectively partition a directed graph. In their work they came up with the idea of defining what they termed as "balance vertices" and showed how to find and use these balance vertices to get a vertex cut of a graph which is balanced. According to them a vertex *v* is said to be a balance vertex of a graph if and only if that vertex is distant from source and sink and at the same time and the vertex is connected to source and sink.

Relying on the fact that a vertex cut that passed through the balance vertices will result into two partitions which are balanced, their algorithm first finds the set of balance vertices that can be used as vertex cut, and the graph is then partitioned in a hierarchical manner by recursively applying structurally balanced cuts.

They formulated the problem and defined two quality measures (metrics) namely *expansion* and *modularity* to evaluate the partitioning solutions.

Expansion is the number of vertices through which the two partitions $P\,i$ and $P\,j$ are split.

They defined it as follows:

$$expansion_{ncut1} = \frac{vertex\_cut(P_i, P_j)}{\min\{|P_i|, |P_j|\}}$$

The above definition is applicable in a situation where only two partitions and $P\,j$ are involved.

It can be extended to handle more than two partitions as defined below.

$$expansion_{ncut1} = \frac{vertex\_cut(P_1, P_2, \ldots P_N)}{\min\{|P_1|, |P_2|, \ldots . |P_N|\}}$$

They observed that the above definition does not account for edge distributions in the resulting clusters. An alternative definition which directly accounts for the edge distribution is given below:

$$expansion_{ncut2} = \frac{vertex\_cut(P_i, P_j)}{\min\{|P_i, E|, |P_j, E|\}}$$

Similarly, this can also be extended to be use in the context in which more than two partitions are involved as defined below.

$$expansion_{ncut2} = \frac{vertex\_cut(P_1, P_2, \ldots P_N)}{\min\{|P_1, E|, |P_2, E|, \ldots . |P_N, E|\}}$$

Where: $|P_i, E|$ is the number of edges in the partition $P\,i$. In this scenario, the size of the vertex-cut is normalized in relation to the sizes of the partitions in terms of their numbers of edges. According to them, like *expansion*, *modularity* is also defined in two ways, i.e either based on the number of vertices or based on the number of edges within a partition. The first definition is given below:

$$modularity_{ncut1} = \sum_{1 \le i \le m} \left(\frac{|V_{i,i}|}{|V|} - \left(\sum_{j \neq i} \frac{|V_{i,j}|}{|V|} \right)^2 \right)$$

Where: $|V_{i,j}|$ denotes the number of vertices in the graph that are common between partitions *Pi* and *Pj* and $|V_{i,i}|$ denotes the number of vertices in partition *Pi*.

Below is the second definition which is based on the number of the edges in the partitions.

$$modularity_{ncut2} = \sum_{1 \le i \le m} \left(\frac{|E_{i,i}|}{|E|} - \left(\sum_{j \neq i} \frac{|V_{i,j}|}{|V|} \right)^2 \right)$$

Whereas before $|E_{i,i}|$ denotes the number of edges in partition *Pi*. According to them, the higher the modularity is the better is the partitioning.

In a similar work by Rahimian [5], they considered the two techniques of partitioning graph (i.e. *vertex-cut* and *edge-cut* partitioning), and proposed fully distributed algorithm called JA-BE-JA. The proposed algorithm is completely parallel, the operation of the algorithm is not coordinated centrally, each vertex is processed independently, and only adjacent neighbors of a vertex and some few random subset of vertices in the graph need to be known locally. They reported that their algorithm computes very low *vertex cut* which are proved to be more efficient than *edge-cut* for processing real world graphs. Their algorithm can be use in both vertex-cut as well as edge-cut partitioning.

In their work they defined two variations of the partitioning problem, namely *edge-cut* and *vertex-cut partitioning.*

In the case of *vertex-cut* partitioning, they consider the following partition metrics to evaluate the quality of partition produced by their partitioner.

Vertex-Cut: This metric finds the number of times that graph vertices are cut. A vertex which is cut once will be replicated in two partitions, and a vertex that is cut twice will be replicated in over three partitions. They added that this

metric is important one because if we want to use the partitioned graph, (for example let us assume we want to run page rank algorithm on the partitioned graph). If a particular vertex is replicated in several partitions, every computation that involves a modification to that vertex, should be propagated to all other replicas of that vertex in order to have consistency. Therefore, *Vertex-Cut* directly affects the cost of communication imposed by the partitioned graph. The more the number of replicas the more the communication overhead. The target is therefore to minimize this metric.

Normalized Vertex Cut: They used this metric to calculate the vertex-cut of the final partitioning relative to random partitioning, thus, it shows to what extent their algorithm can reduce vertex-cut.

Standard Deviation of Partition Sizes: They used this metric to measure the Standard Deviation of normalized size of the partitions. To be more precise, they first measured the size of the partitions, in terms of number of edges, relative to average size. They added that in a perfect balanced partition, the normalized size should be 1. They then calculated how much the normalized size deviates from 1.

Chen [12] proposed a new hybrid vertex-cut partitioning algorithm that uses differentiated partitioning for low and high degree vertices. They observed that existing vertex cut algorithms normally target at reducing the replication factor of all vertices, however, according to them the key is, instead of reducing replication factor of low degree vertices, since high degree vertices inevitably need to be replicated in most of the partitions. Majority of current vertex cut heuristics have a bias towards high degree vertices while paying little attention to low degree vertices. They proposed a balanced p-way hybrid-cut that focuses on reducing replication factor of low degree vertices. Their algorithm uses differentiated partitioning to low degree and high degree vertices. In their work they targeted minimizing *replication factor* metric while maintaining the *balance* metric. In addition to these two metrics they also used *communication cost* metric.

These three metrics are already defined in Mykhailenko [4] which we reported earlier.

5.2. Edge Cut Metrics

In this sub-section we present our findings from papers that talked about *edge-cut* partitioning.

The study of partitioning a billion-node problem can be seen in the works of Wang [1] and [13]. Wang [1] proposed a multilevel propagation partitioning technique that can partition billion-node graph within some hours on distributed cluster consisting of several nodes. They evaluated the quality of the partition produced by their approach using the following metrics: Size of *edge cut* denoted by *EC* defined as follows

$$\mathrm{EC(P)} = \sum \mathrm{EC(v)}$$

where *EC* is the number of neighbors of *v* that are not part of *v*'s partition. Their goal is to minimize this metric, because according to them in distributed systems moving along cut edges means doing remote access, and too much of that results in costly communication overhead. By minimizing this metric, communication overhead will be reduced which will result in the reduction of the execution time.

Communication volume of Partitions denoted as *CV* defined as follows:

$$\mathrm{CV(P)} = \sum \mathrm{CV(v)}$$

where *CV* denotes the number of partitions excluding *v*'s partition that contains the neighbors of *v*. Their goal is to minimize this metric. According to them this metric is important because in some cases the total number of cross-partition edges may not be the goal to minimize. Example in BSP, for each loop, individual messages between two machines are assembled into a single message, which will incur a single network communication. This makes sense because the communication cost most of the times comes from the number of network communications not size of the individual messages. Therefore, instead of minimizing the total number of edges that crossed partition, there is also a need to minimize total communication volume.

Balance: They also measured the quality of a partition by its balance, a partition *P* is balanced if each partition has almost the same number of vertices. This metric is defined for given k partitions. It is expected the graph should be distributed equally among the partitions. i.e each partition is expected to approximately has $\frac{|V|}{k}$ vertices. But generally relaxation is allowed so that the number of vertices in a single partition is $(1 \pm \epsilon)\frac{|V|}{k}$ with $0 < \epsilon \ll 1$.

Meyerhenke [14] developed a new fast method for the improvement of graph partitioning. On how to measure the quality of a partition they reported that this depends on the application in most cases. Beside the popular *edge-cut* metric (denoted by *ext* in their work), they also measured total number of *boundary node* (which they denoted as *bnd*) which is used to measure communication costs. To access the partition shapes, they also included the results on the partition *diameter* (denoted by *diam*).

For a given partitioning *P*, they defined the metrics as:

Cut edges:

$$(p) = |e = \{u, v\} \in E : \Pi(u) = p \wedge \Pi(v) \neq v|$$

Boundary nodes:

$$(p) = |v \in V : \Pi(v) = p \wedge \exists \{u, v\} \in E : \Pi(u) \neq p|$$

According to them this metric is important because it measures communication costs in parallel numerical simulations more accurately.

Diameter:

$$(p) = \max \{d(u, v) = \Pi(u) = \Pi(v) \neq v\}$$

According to them this metric is important because for some applications, the shape of the partitions, in particular small aspect ratios but connectedness and smooth boundaries play an important role.

Hendrickson [15] observed that *Edge Cut* metric used by

standard graph partitioning methodology is a wrong one, according to them it lacks expressibility, it is only approximation of the total communication volume. They suggested a more appropriate metric *Boundary Cut*, which is number of external partitions in which vertex *V*i has neighbors. It is defined as

$$boundary\ cut = \sum bi$$

Where *bi* is the number of external partitions in which vertex *vi* has neighbors.

Tsourakakis [16] proposed a unifying framework for graphs partitioning and they evaluated their algorithm by measuring two quantities from the resulting partitions. In particular, for a fixed partitioning P they used the measures of the fraction of edges cut denoted by λ and the normalized maximum load denoted by ρ defined as follows:

$$\lambda = \frac{\#\ edges\ cut\ by\ p}{\#\ total\ edges}$$

$$\rho = \frac{maximum\ load}{\frac{n}{k}}$$

Martella [17] proposed graph partitioning algorithm which is a scalable and also adaptive based on label propagation. They evaluated the quality of the partitions produced by their algorithm based on two metrics which are *locality* and *balance*. They measured locality as the ratio of local edges denoted by ϕ and balance as the maximum normalized load denoted by ρ defined as follows:

$$\phi = \frac{\#\ local\ edges}{|E|}$$

$$\rho = \frac{maximum\ load}{\frac{|E|}{k}}$$

Where k denotes the number of partitions, *# local edges* denotes the number of edges that connect vertices which are assign to the same partition, and *maximum load* is the number of edges assigned to the partition which is most loaded. The maximum normalized load metric is used to measure unbalance and represent the difference in percentage of the partition which is most loaded from a perfectly balanced one.

According to them locality is an important metric because typically, graph processing systems vertices are distributed across machines, and because communication in such systems is as a result of graph edges, network traffic normally is generated when edges cross partition boundaries. Therefore, an efficient partitioning algorithm should minimize the number of edges that cross partition boundaries. Similarly, balance is important metric because partitions that balance loads improve processing latency and resource utilization.

Martella [17] highlighted that the ability to maintain local edges depends on the number of partitions. Intuitively, the more partitions, the harder it is to maintain locality. In their experiment they varied the number of partitions and measured locality and balance for different graphs.

Figure 2 summarized the metrics used in the case of vertex-cut approach.

S/N	Ref.	Metric name	Definition	Objective
1	[4]	Balance	$Bal = \frac{max_{i=1\ldots N}(\lvert E_i\rvert)}{\lvert E\rvert/N}$	
2	[4]	Normalized standard deviation	$NSD = \sqrt{\Sigma\left(\frac{\lvert Ei\rvert}{E/N} - 1\right)^2 \frac{1}{N}}$	Minimize
3	[4]	Vertex-cut	$VC = \lvert V\rvert - \sum_{i=1}^{N} \lvert\overline{F}(E_i)\rvert$	Minimize
4	[4]	Communication cost	$CC = \sum_{i=1}^{N} \lvert F(E_i)\rvert$	Minimize
5	[4]	Replication factor	$RF = \sum_{i=1}^{N} \lvert V(E_i)\rvert \frac{1}{\lvert V\rvert}$	Minimize
6	[4]	Largest partition	$LP = max_{i=1\ldots N}\lvert V(E_i)\rvert$	
7	[11]	$expansion_{ncut1}$	$expansion_{ncut1} = \frac{vertex\_cut(P_1, P_2, \ldots P_N)}{\min\{\lvert P_1\rvert, \lvert P_2\rvert, \ldots . \lvert P_N\rvert\}}$	Minimize
8	[11]	$expansion_{ncut2}$	$expansion_{ncut2} = \frac{vertex\_cut(P_1, P_2, \ldots P_N)}{\min\{\lvert P_1, E\rvert, \lvert P_2, E\rvert, \ldots . \lvert P_N, E\rvert\}}$	Minimize
9	[11]	$modularity_{ncut1}$	$modularity_{ncut1} = \sum_{1\leq i\leq m}\left(\frac{\lvert V_{i,i}\rvert}{\lvert V\rvert} - \left(\sum_{j\neq i}\frac{\lvert V_{i,j}\rvert}{\lvert V\rvert}\right)^2\right)$	Minimize
10	[11]	$modularity_{ncut2}$	$modularity_{ncut2} = \sum_{1\leq i\leq m}\left(\frac{\lvert E_{i,i}\rvert}{\lvert E\rvert} - \left(\sum_{j\neq i}\frac{\lvert V_{i,j}\rvert}{\lvert V\rvert}\right)^2\right)$	Minimize
11	[5]	Vertex-cut	$VC = \lvert V\rvert - \sum_{i=1}^{N} \lvert\overline{F}(E_i)\rvert$	Minimize
13	[5]	Standard Deviation of Partition sizes	$NSD = \sqrt{\Sigma\left(\frac{\lvert Ei\rvert}{E/N} - 1\right)^2 \frac{1}{N}}$	Minimize

Figure 2. *Evaluation metrics for vertex-cut approach.*

Figure 3 summarized the metrics used in the case of edge-cut approach.

S/N	Ref.	Metric name	Definition	Objective
1	[13]	Edge-cut	$EC(P) = \sum EC(v)$	Minimize
2	[13]	Communication Volume	$CV(P) = \sum CV(v)$	Minimize
3	[13]	Balance	$Balance = (1 \pm \epsilon)\frac{\lvert V \rvert}{k}$	
4	[14]	Edge-cut	$ext(p) = \lvert e = \{u, v\} \in E : \Pi(u) = p \wedge \Pi(v) \neq v \rvert$	Minimize
5	[14]	Boundary nodes	$bnd(p) = \lvert v \in V : \Pi(v) = p \wedge \exists \{u, v\} \in E : \Pi(u) \neq p \rvert$	Minimize
6	[14]	Diameter	$diam(p) = \max\{dist(u, v) = \Pi(u) = \Pi(v) \neq v\}$	
7	[15]	Boundary cut	$boundary\ cut = \sum bi$	Minimize
8	[16]	Edge-cut	$\lambda = \frac{\#\ edges\ cut\ by\ p}{\#\ total\ edges}$	Minimize
9	[16]	Normalized Maximum load	$\rho = \frac{maximum\ load}{\frac{n}{k}}$	
10	[17]	Locality	$\phi = \frac{\#\ local\ edges}{\lvert E \rvert}$	Minimize
11	[17]	Balance	$\rho = \frac{maximum\ load}{\frac{\lvert E \rvert}{k}}$	

Figure 3. *Evaluation metrics for edge-cut approach.*

6. Conclusion

In this research large body of published papers about graph partitioning were studied. Our finding shows that very few work was done about *vertex-cut* partitioning. Majority of published work focus on *edge-cut* partitioning. We also found that many authors rely on using *execution metrics* to evaluate their partitioners by performing some comparisons with state of the art partitioners. Very few authors rely on *partition metrics*. There is still need to investigate and define more metrics, and there is also need to adapt the metrics used in the context of *edge-cut* portioning so that they will also be applicable in the context of the *vertex-cut* partitioning.

References

[1] L. Wang, Y. Xiao, B. Shao, and H. Wang, "How to partition a billion-node graph," in *2014 IEEE 30th International Conference on Data Engineering*, 2014, no. 20100071120032, pp. 568–579.

[2] A. Ching, S. Edunov, M. Kabiljo, D. Logothetis, and S. Muthukrishnan, "One trillion edges," *Proc. VLDB Endow.*, vol. 8, no. 12, pp. 1804–1815, Aug. 2015.

[3] A. Kyrola, G. Blelloch, and C. Guestrin, "GraphChi," *USENIX Symp. Oper. Syst. Des. Implement.*, vol. 10, no. 31, pp. 31–46, 2012.

[4] H. Mykhailenko, G. Neglia, and F. Huet, "Which metrics for vertex-cut partitioning?," in *2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)*, 2016, pp. 74–79.

[5] F. Rahimian, A. H. Payberah, S. Girdzijauskas, M. Jelasity, and S. Haridi, "A Distributed Algorithm for Large-Scale Graph Partitioning," *ACM Trans. Auton. Adapt. Syst.*, vol. 10, no. 2, pp. 1–24, Jun. 2015.

[6] H. Mykhailenko, F. Huet, and G. Neglia, "Comparison of Edge Partitioners for Graph Processing," in *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2016, pp. 441–446.

[7] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "PowerGraph: Distributed Graph-Parallel Computation on Natural Graphs," in *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*, 2012, vol. 12, pp. 17–30.

[8] H. P. Sajjad, A. H. Payberah, F. Rahimian, V. Vlassov, and S. Haridi, "Boosting Vertex-Cut Partitioning for Streaming Graphs," in *2016 IEEE International Congress on Big Data (BigData Congress)*, 2016, pp. 1–8.

[9] C. Xie, W.-J. Li, and Z. Zhang, "S-PowerGraph: Streaming Graph Partitioning for Natural Graphs by Vertex-Cut," Nov. 2015.

[10] Spark, "GraphX Programming Guide," *GraphX - Spark 3.2.0 Documentation*. [Online]. Available: https://spark.apache.org/docs/latest/graphx-programming-guide.html. [Accessed: 27-Jan-2022].

[11] M. Kim and K. S. Candan, "SBV-Cut: Vertex-cut based graph partitioning using structural balance vertices," *Data Knowl. Eng.*, vol. 72, pp. 285–303, Feb. 2012.

[12] R. Chen, J. Shi, Y. Chen, B. Zang, H. Guan, and H. Chen, "PowerLyra," *ACM Trans. Parallel Comput.*, vol. 5, no. 3, pp. 1–39, Jan. 2019.

[13] D. Zheng, D. Mhembere, R. Burns, J. Vogelstein, C. E. Priebe, and A. S. Szalay, "FlashGraph: Processing Billion-Node Graphs on an Array of Commodity SSDs," *Proc. 13th USENIX Conf. File Storage Technol. FAST 2015*, pp. 45–58, Aug. 2014.

[14] H. Meyerhenke, B. Monien, and T. Sauerwald, "A new diffusion-based multilevel algorithm for computing graph partitions of very high quality," in *2008 IEEE International Symposium on Parallel and Distributed Processing*, 2008, vol. 69, no. 9, pp. 1–13.

[15] B. Hendrickson and T. G. Kolda, "Graph partitioning models for parallel computing," *Parallel Comput.*, vol. 26, no. 12, pp. 1519–1534, Nov. 2000.

[16] C. Tsourakakis, C. Gkantsidis, B. Radunovic, and M. Vojnovic, "FENNEL," in *Proceedings of the 7th ACM international conference on Web search and data mining*, 2014, pp. 333–342.

[17] C. Martella, D. Logothetis, A. Loukas, and G. Siganos, "Spinner: Scalable Graph Partitioning in the Cloud," in *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, 2017, vol. 26, no. 12, pp. 1083–1094.