A Graph Partitioning Algorithm for Edge or Vertex Balance

Adnan El Moussawi[1], Nacéra Bennacer Seghouani[1], and Francesca Bugiotti[1]

Laboratoire de Recherche en Informatique LRI, 91190 Gif-sur-Yvette, France
`{firstname.lastname}@lri.fr`
`http://www.lri.fr`

Abstract. The definition of effective strategies for graph partitioning is a major challenge in distributed environments since an effective graph partitioning allows to considerably improve the performance of large graph data analytics computations. In this paper, we propose a multi-objective and scalable Balanced GRAph Partitioning (B-GRAP) algorithm to produce balanced graph partitions. B-GRAP is based on Label Propagation (LP) approach and defines different objective functions to deal with either vertex or edge balance constraints while considering edge direction in graphs. The experiments are performed on various graphs while varying the number of partitions. We evaluate B-GRAP using several quality measures and the computation time. The results show that B-GRAP (i) provides a good balance while reducing the cuts between the different computed partitions (ii) reduces the global computation time, compared to Spinner algorithm.

Keywords: Large graph partitioning, vertex balance, edge balance, parallel processing.

1 Introduction

In recent years, large-scale graph analytics and mining have been widely used in various domains such as communication network, urban transportation, biological data and social networks. In this context the efficient processing of large graphs becomes a new challenging task. Many research works focused on graph-based parallel computation algorithms in distributed systems [?,?,?]. The distribution of the workloads on several machines helps to reduce the overhead computation time. However, this distribution requires multiple exchanges of messages between the machines with a typically high cost.

Graph Partitioning (GP) algorithms have taken a lot of attention in recent decade as a key prerequisite for an efficient processing and many works focused on this problem [?,?,?,?]. An efficient partitioning algorithm allows to minimize the total computation cost while a good balanced load makes better leverage of the entire system. The GP problem aims to divide the graph into a given number of partitions while minimizing the number of their inter-connecting edges (called *cuts*) and balancing their sizes w.r.t. the number of vertices or the number of edges. An *edge-balanced* GP divides the edges of the graph into nearly equal sized partitions. In contrast, a *vertex-balanced* GP divides the vertices of the graph into equisized partitions. Each objective has its own advantage. For a graph analysis task that needs few communications between vertices, the *vertex-balanced* GP is more beneficial. On the contrary, for task where the vertices exchange messages frequently, balancing the number of edges has more advantage.

Different GP partitioning strategies approaches have been studied in the literature. Multilevel GP approaches [**?**] defined a well *vertex-balanced* partitioning algorithm which shown high quality in terms of *cuts*. But it requires high resource usage and computation time and it does not scale with large graphs. Streaming GP methods [**?**,**?**] used an online graph partitioning, by considering *vertex-balance* or *edge-balance* constraints which reduces the overhead computation comparing to multilevel approaches. But on the other hand, the results of such methods are of less quality and depend on the order of vertex or edge processing. Moreover the partitioning is not adaptive to the graph's changes. Recent works have taken advantage of the lightweight mechanism of Label Propagation (LP) approach to improve the partitioning process [**?**,**?**,**?**]. In [**?**,**?**], the authors used LP to coarsen the graph in a multilevel partitioning approach while balancing the vertices. [**?**] extended LP to compute the entire partitioning basing on Giraph [**?**] programming model, while considering only *edge-balance* constraint.

In this paper we propose a new multi-objective and scalable *B*alanced *GRA*ph *P*artitioning algorithm (B-GRAP), based on LP approach, to produce balanced graph partitions. Our main contributions are:

– An optimized partitioning initialization that helps to improve the propagation of labels and to reduce the computational overhead comparing to similar approaches.
– A new scalable and parallel partitioning algorithms B-GRAP$_{\mathrm{VB}}$ and B-GRAP$_{\mathrm{EB}}$ that respectively address the vertex balance and the edge balance problems on both directed and undirected graphs.
– We implement our algorithm on top of the open source distributed graph processing system Giraph [**?**]. This allows us to take advantage from the parallel processing architecture in order to effectively parallelize B-GRAP.
– The evaluation of B-GRAP, using different measures (quality and time) on heterogeneous real-worlds and synthetic graphs, shows good performance while scaling with the number of partitions and size of a graph.

This paper is structured as follows. In Section 2, we detail some approaches related to graph partitioning problem. In Section 3, we present LP approach and B-GRAP main notations. In Section 4, we define B-GRAP, its initialization, the propagation functions for vertex or edge balance, then the measures we use to evaluate the quality of the partitioning are presented in Section 5. In Section 6, we provide the experimental study conducted in order to evaluate B-GRAP. Finally, in Section 7, we give our conclusions and future perspectives.

2 Related Work

During the last decade, research communities working on graph datasets have given a lot of interest to the definition of new strategies for large graph parallel computing and analytics in a distributed environment. This context opened up new challenges to define efficient graph partitioning algorithms [?,?,?,?,?]. One of the main challenges consists in defining graph partitioning algorithms that allow to balance the workload among the nodes of a distributed computing environment and to reduce, at the same time, the communication load over the network.

A common strategy in large graph partitioning is to use multilevel approaches [?]. The idea is to generate a first partition on the basis of a reduced view of the graph in which a vertex represents many vertices of the original graph. For example a triangle of three vertices can be reduced to one. The algorithms then expands the graph taking into account the whole initial graph. This family of approaches alternates three main phases: (i) coarsen the graph by collapsing adjacent vertices satisfying some matching criteria, (ii) partition the coarsened graph using any partitioning algorithm, (iii) the un-coarsening or refinement, which means generalizing the partition from last phase by mapping back the results to the original graph. METIS [?] is one of the multilevel graph partitioning algorithm family. This algorithm is known for its ability to produce partitioning with high quality w.r.t. the number of cuts, but with the disadvantage of the high computation time to obtain several intermediate results. Another known multilevel graph partitionner is Scotch [?] which deals with the graph changes and does not require to start the partitioning from scratch, in contrast to METIS. The parallel version of both algorithms, ParMETIS [?] and Pt-Scotch [?], show good cuts quality but their performance scales poorly with respect to the number of processors as shown in [?].

During the last years stream graph partitioning has been proposed in order to reduce the complexity [?] of multilevel approaches, since they take into account the entire input graph during the whole computation. These algorithms assign edges and vertices to various partitions by running a single pass through the whole graph. The goal, of the most part of these algorithms, is to guarantee the edge balance [?,?] and to find a partitioning that reduces the usage of the resources and the computation overhead. These methods are faster than multilevel algorithms but they build partitioning with lower quality, in term of cuts, due to the sensitivity to the stream order. Moreover, it's generally difficult to parallelize streaming algorithms.

Other works have used the label propagation approach (LP) [?] to partition large graphs. LP was mainly used for community detection in social networks [?,?]. Making use of LP for the graph partitioning problem was motivated by the lightweight mechanism that uses the network structure to guide its progress. LP partitioning methods generate less intermediary results than multilevel approaches, which need to store many intermediate results such as the coarser graph, and run with a lower complexity. Furthermore, LP method is semanticaware, given the existence of local closely connected substructures, a label tends to propagate within such structures.

The authors of [?] propose ParHIP, a distributed memory parallel partitioning algorithm, that takes advantage of both multilevel and LP approaches. The authors adapt and parallelize LP technique for both coarsening and refinement step, using the Message Passing Interface (MPI), while considering the vertex-balance constraint. Their experimental results show that ParHIP is more scalable and achieves higher quality than existing state of the art methods like ParMETIS and PT-Scotch.

Finally, in [?] the authors define a distributed partitioning algorithm called Spinner that considers only edge balance. Spinner is based on LP approach and runs on the top of Giraph API [?]. Compared to the previous work, Spinner supports the parallelism and can adapt an existing partitioning to consider graph updates by adding or removing vertices and edges and changing the number of partitions. The algorithm divides N vertices across K partitions, while trying to keep similar the number of edges in each partition.

In this paper we present a new algorithm for balanced graph partitioning based on LP approach and using Giraph programming model. Compared to the literature, our algorithm B-GRAP deals with edge-based or vertex-based balanced partitioning while decreasing the number of cuts and computation time. Moreover, B-GRAP defines an initialization heuristic which allows to improve the propagation of labels across the graph and to accelerate the convergence of the algorithm on large graphs.

3 Preliminaries

Given a number of partitions K, a directed graph $G = \langle V, E, \omega \rangle$, where V is a set of vertices and E a set of weighted edges with $\omega : E \to \mathbb{R}^+$. Let $L = \{l\}_{l=1}^{K}$ be a set of partition labels defined by a labeling function $\phi : V \to L$ such that $\phi(v) = l$ means that v belongs to the partition with label l. The naïve LP algorithm proceeds as follows. Initially, a unique label l_v is assigned to each vertex v. Then, the label of each $v \in V$ is propagated and updated iteratively to its neighborhood $N(v) = \{u \in V | (v,u) \vee (u,v) \in E\}$ and is updated until a given convergence criteria is reached. The label updating is done by taking into account the most frequent label among $N(v)$ labels. More formally, let $\mathcal{F}_{\mathrm{LP}(v,l)}$ be the frequency of a label l in the neighborhood of v, defined by:

$$\mathcal{F}_{\mathrm{LP}(v,l) = \sum_{u \in N(v)} \omega(v,u) \delta\left(\phi(u), l\right)} (1)$$

where $\phi(u)$ gives the current label of u and δ is the Kronecker delta function, which equals 1 if $\phi(u) = l$, and 0 otherwise. The label of vertex v is replaced by the label that maximizes the frequency function:

$$l_v = \underset{l}{\operatorname{argmax}} \ \mathcal{F}_{\mathrm{LP}(v,l)} (2)$$

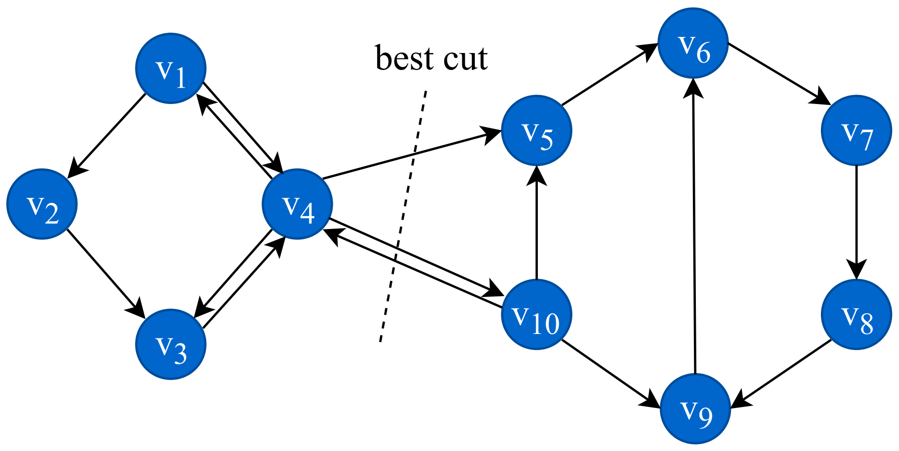

Fig. 1: Vertex-balanced and edge-balanced 2-partitioning graph example

If many maximal labels exist and do not include the current label of v, one of them is randomly chosen. LP algorithm stops if $\sum_{v \in V} \sum_{l \in L} \mathcal{F}_{\mathrm{LP}(v,l)}$ converges according to a given threshold ϵ.

We note that naïve LP algorithm does not take into account the directions of edges. To consider directed graphs, virtual edges are added such that: $\forall (v,u) \in E, \omega(v,u) = 2$ and if $(u,v) \notin E, (u,v)$ is added with $\omega(u,v) = 1$ which we call *virtual edge*. Note that the Giraph data model is a distributed directed graph, where every vertex is aware of its outgoing edges only, but not of the incoming ones. Adding virtual edges in this case, allows to a vertex to discover its entire neighborhood $N(v)$, while the weight allows to consider the direction as well as to distinguish these virtual edges added.

4 B-GRAP algorithm

Our goal is to define a K-balanced and LP-based partitioning algorithm that decreases the total cuts while considering the vertex balance or the edge balance constraints in directed graphs.

To illustrate our objectives we consider the example provided in Figure 1. This example presents a small graph of 10 vertices and 16 directed edges. We would divide the graph into two balanced partitions, using either a vertex-balanced partitioning or the edge-balanced partitioning. First, we note that the best 2-partitioning that minimizes the edge cuts is $P_1 = \{v_1, v_2, v_3, v_4\}$ and $P_2 = \{v_5, v_6, v_7, v_8, v_9, v_{10}\}$, where the cuts $(v_4, v_5), (v_4, v_{10}), (v_{10}, v_4)$. To achieve a vertex-balanced partitioning, a vertex from P_2 should be moved to P_1, while caring about the cuts. In this case, moving v_{10} to P_1 is the most advantageous because it introduces less edge cuts. For the edge-balanced partitioning, no change is needed. Indeed, both partitions holds $|E|/2$ *directed edges* and the number of edge cuts is minimized. Note that if the directness of edges is ignored, this partitioning remains unbalanced. In fact, in an undirected graph, each edge is considered to be bidirectional, as a result the number of edges in P_1 is 12 and 16 in P_2.

In the following, we present in detail B-GRAP algorithm. First, we present the initialization strategy, then we define the update functions $\mathcal{F}$ to build vertex (or edge) balanced partitions, and finally we present the measures used for the evaluation of partitioning quality.

4.1 Initialization optimization:

To improve the performance of propagation approach in our algorithm, we define an initialization strategy, called B-GRAP$_{init}$, which considers only *hub* vertices having a high outgoing degree $d^+(.)$. The intuition behind this choice is that the higher $d^+(v)$, the more $\phi(v)$ will be propagated and considered as frequent label. This differs from LP approach that considers all the vertices. As a result, the candidates to be considered as frequent labels (i.e. the labels with high probability to be the most frequent) are propagated faster at the first *partial propagation* iteration. The initialization we defined should guarantee a faster label propagation and smaller number of exchanges between vertices given the fact that the nodes having the higher probability to be selected to propagate their label have been already initialized.

B-GRAP is described in Algorithm 1. Let τ be a given minimum out degree threshold to consider that a vertex v as a *hub* vertex. The algorithm proceeds as follows. First, we initialize the set of labels L (Line 1). Then, each $v \in V$, such $d^+(v) > \tau$ is assigned a random label $\in L$ and those labels are propagated to neighbors (Line 2). Then, the label of these neighbors are updated and propagated iteratively using an update function (Lines 4-7). The vertices are then checked and those not reached by the update/propagation step are initialized randomly, to ensure that all vertices are assigned a label (Line 9). The algorithm repeats the update/propagate step until convergence (Line 10).

4.2 Balanced partitioning:

In the basic LP partitioning, the label update is done without caring about the size of the partitions. Consequently, this can lead to an unbalanced partitioning. Moreover the update function of LP (Eq. 1) has a trivial optimal solution that consists of assigning all vertices to a single label, i.e. to a single partition. A standard resolution approach to deal with such a problem is to integrate the balance constraints to the update function via a penalty function. The LP update function becomes:

$$\mathcal{F} = \mathcal{F}_{\mathrm{LP}} + \lambda \mathcal{P} \quad (3)$$

where $\mathcal{P}$ represents penalty terms and λ is a weight parameter. In B-GRAP algorithm, we define two update functions $\mathcal{F}_{\mathrm{VB}}$ and $\mathcal{F}_{\mathrm{EB}}$ which respectively deal with vertex and edge balance constraints.

Algorithm 1 B-GRAP

```
input  $G = \langle V, E, w \rangle$, $K$, $\tau$, $\epsilon$
output  a partitioned graph $G = \langle V, E, w, L \rangle$
 1: Initialize the set of labels $L = \{l\}_{l=1}^{K}$
 2: for $\big(v \in V, d^{+}(v) > \tau\big)$ initialize $\phi(v)$ randomly from $L$ and propagate to $N(v)$
 3: repeat
 4:     {Search frequent labels}
 5:     for $(v \in V, l \in L)$ get the set of frequent labels w.r.t an update function
 6:     {Update and propagate}
 7:     Update and propagate $\phi(v)$ to $N(v)$
 8:     {Check unassigned vertices}
 9:     for $\big(v \in V,\ \phi(v) = \emptyset\big)$ initialize $\phi(v)$ randomly from $L$ and propagate to $N(v)$
10: until $\Delta\big(\mathcal{F}_{\mathrm{LP}(G,L)}\big)_{\le \epsilon}$
11: return $G = \langle V, E, w, L = \{\phi(v)\}_{v \in V} \rangle$
```

Vertex balance: Given a directed graph $G = \langle V, E, \omega \rangle$, a vertex-balanced partitioning divides the vertices into disjoint partitions of nearly equal size, while minimizing the number of edge cuts between partition. Let $size(V, l)$ be number of vertices having l as label, $size(V, l) = |\{v \in V \mid \phi(v) = l\}|$.

In a perfect balanced partitioning, the size of each partition should be equal to $|V|/K$. In other words, the distribution of vertices in the partitions should be close to a uniform distribution $\mathbf{U} = \langle 1/K, \ldots, 1/K \rangle$, where $1/K$ is called the balance factor. To handle the balance between the partitions, we define vertex-balance $\mathcal{P}_{\mathrm{VB}}$ penalty function that penalizes $\mathcal{F}$ when trying to assign a vertex to a partition violating the balance constraints as follows:

$$\mathcal{P}_{\mathrm{VB}(l) = \frac{1}{K} - \frac{size(V,l)}{|V|}} \quad (4)$$

This function measures the difference between the balance factor $1/K$ and the ratio of vertices assigned to l label. The larger the ratio of vertices with label l is, the higher the penalty to update the vertex label with l is.

At this stage, the number of edge cuts between the partitions is not considered. Thus, a vertex could move to a partition that increases the edge cuts. Given a vertex v and label l, we define a second penalty function as follows:

$$\mathcal{P}_{\mathrm{EC}(v,l) = \frac{|cut(v,l)|}{d^{+}(v)}} \quad (5)$$

where $cut(v, l) = \{(v, u) \in E \mid \phi(u) = l\}$ is the set of edges outgoing from v to vertices in a partition with label l. This function measures the ratio of cuts which penalizes a vertex v to move to a partition with l label if the number of its outgoing edges to this partition is low (normalized to the out degree of v). Thus, when a vertex has more connections to a partition than to the others, the penalty gives more advantage to move to this partition and vice versa. By considering the penalty functions defined in Eq. 4 and Eq. 5, the vertex balance update function is defined in the following equation:

$$\mathcal{F}_{\mathrm{VB}(v,l)=n\mathcal{F}_{\mathrm{LP}+\lambda\left(\kappa\mathcal{P}_{\mathrm{EC}(v,l)+(1-\kappa)\mathcal{P}_{\mathrm{VB}(v,l)}}\right)}} \tag{6}$$

where n is a normalization constant equal to $\frac{1}{\sum_{u\in N(v)}\omega(v,u)}$. The balance factor $\frac{1}{K}$ could be omitted as it is constant, in this case $\mathcal{P}_{\mathrm{VB}}$ variate $\in [0..1]$. The parameter κ is a weight ranging between 0 and 1 which gives more or less importance to balance penalty against the edge cuts penalty. We set κ to 0.5 by default.

Edge balance: An edge-balanced partitioning divides the graph into disjoint partitions holding nearly equal number of edges, while minimizing the number of edge cuts between partition. Let $size(E,l)$ be the number of outgoing edges from a partition with label l, $size(E,l) = \sum_{v\in V,\phi(v)=l} |d^{+}(v)|$. Similarly to the vertex-balance partitioning, we define the following edge-balance penalty function:

$$\mathcal{P}_{\mathrm{EB}(l)=\frac{1}{K}-\frac{size(E,l)}{|E|}} \tag{7}$$

This function discourages a vertex move to a partition with l label, when the ratio of edges in the partition l is closer or larger than the balance factor. Comparing to vertex balance, edge balance maximizes the edge locality in each partition, which contributes to minimizing the edge cuts. Thus, there is no need to add additional penalty to the update function as defined in Eq. 6. The edge-balance update function is formulated as follows:

$$\mathcal{F}_{\mathrm{EB}(v,l)=n\mathcal{F}_{\mathrm{LP}+\lambda\mathcal{P}_{\mathrm{EB}(l)}}} \tag{8}$$

We note that Spinner algorithm [?] (see Section 3) uses the normalized unbalance as penalty function. Comparing to Eq. 7, the edge-size of a partition is normalized by the size of a perfect balanced partition, i.e. $\frac{|E|}{K}$. Moreover, their penalty function that measures the edge balance for each partition, considers both virtual and real edges. The function we defined in Eq. 8 considers only real edges.

5 Partitioning Evaluation Measures

To evaluate the quality of the partitioning produced by our algorithm B-GRAP, we use two standard measures: the ratio of edge cuts **EC** and the Jensen Shannon divergence (**JSD**) [?].

The edge cuts ratio is the ratio of edges connecting each two vertices in two different partitions w.r.t the total number of edges.

$$\mathbf{EC} = \frac{\sum_{v\in V}\sum_{l=1}^{K}|cut(v,l)|}{|E|} \tag{9}$$

The Jensen Shannon divergence (JSD) is the symmetric version of the KullbackLeibler divergence known as a standard measure to compute the divergence between two distributions. This is a symmetric measure varying in the interval $[0 \dots 1]$, where a value close to 0 indicates that the distributions are similar. Let $\mathbf{P} = \langle p_1, \dots, p_K \rangle$ and $\mathbf{Q} = \langle q_1, \dots, q_K \rangle$ two distributions with the same size. The **JSD** divergence is computed as follows:

$$\mathbf{JSD}(\mathbf{P} \| \mathbf{Q}) = \frac{1}{2}\Big(D_{\mathrm{KL}}(\mathbf{P} \| M) + D_{\mathrm{KL}}(\mathbf{Q} \| M)\Big) \tag{10}$$

$$\text{with } D_{\mathrm{KL}}(\mathbf{P} \| \mathbf{Q}) = \sum_{l=1}^{K} p_l \log(\frac{p_l}{q_l}) \quad \text{and } M = \frac{1}{2}\big(\mathbf{P} + \mathbf{Q}\big)$$

In our case, $\mathbf{P}$ represents the distribution of vertices (or edges) on the partitions, where p_l is the ratio of vertices (or edges) in the partition with label l, and $\mathbf{Q}$ equals to the uniform distribution $\mathbf{U}$. The **JSD** considers the balance of the whole partitioning, comparing to other measures such the maximum normalized unbalance metric (**MNU**) [**?**]. This last used to measure unbalance and represents the percentage-wise difference of only the largest partition from a perfectly balanced partition.

$$\begin{aligned} \mathbf{MNU}_{\mathrm{VB}} &= \frac{\max(|V_l|)}{|V|/K}, \\ \mathbf{MNU}_{\mathrm{EB}} &= \frac{\max(|E_l|)}{|E|/K}, \quad \text{with } l \in L. \end{aligned} \tag{11}$$

Finally, it is important to notice that for **EC**, **JSD**, and **MNU** we consider the directed edges in the original input graph. The virtual edges added for neighborhood discovery (see Section 3) are note taken into account.

6 Experiments

We achieve different experiments on different graph data sets in order to evaluate the quality of edge and vertex-balanced partitioning using **EC**, **JSD** and **MNU** measures defined previously. We compare our approach to Spinner [**?**] because it has shown better results comparing to some existing algorithms. Moreover, since both B-GRAP and Spinner are developed using Apache Giraph environment, we can also provide an evaluation in the same system conditions.

In the following, we first describe the data sets and the experiment settings. Then, we present in detail the results of B-GRAP$_{\mathrm{VB}}$ and B-GRAP$_{\mathrm{EB}}$, compared against Spinner, and achieved on nine graphs.

6.1 Data sets description and experiment settings:

All the experiments are done on a Hadoop cluster of 8 machines, with 64GB RAM and 8 compute cores. B-GRAP algorithm is implemented in Java using Apache

Graph	WikiTalk (W)	BerkeleyStanf (B)	Flixster (F)	DelaunaySC (D)	Pokec (P)	LiveJournal (L)	Orkut (O)	Graph500 (G)	SK-2005 (S)
Directed	yes	yes	yes	yes	yes	yes	yes	no	yes
$\|V\|$	2.4M	0.7M	2.5M	8.4M	16M	4.8M	2.7M	4.6M	50.6M
$\|E\|$	5M	7.6M	7.9M	25.2M	30.1M	69M	117.2M	258.5M	1.9B
Source	[?]	[?]	[?,?]	[?,?]	[?]	[?]	[?]	[?]	[?]

Table 1: Data sets description

Giraph environment [?]. Giraph is an open source implementation of distributed programming framework Pregel [?], designed for Google cluster architecture, with several performance improvement like multi-threading and memory usage optimization. It's built on Hadoop infrastructure to make distributed graph processing and can work with many data storage system supporting graph data (Neo4j, DEX, RDBMS, etc.). In Giraph, the graph is randomly partitioned on several workers (machines)after a complete in-memory load. As in Pergel, Giraph uses a vertex-centric approach to deal with large scale graph processing. In their approach, the computation of the user defined function is done locally, i.e. on each vertex, and in parallel. A vertex contains information about itself and its outgoing edges, it can change its state and the state of these edges by exchanging messages with other vertices at the same iteration, called *super-step*.

In our experiments, we use nine graph data sets of different degree distributions and different sizes in terms of edge and vertex number as summarized in Table 1. Wikitalk (W), Pockec (P), Flixster (F), LiveJournal (L) and Orkut (O) are social online networks graphs. BerkeleyStanf (B) is the berkely.edu and stanford.edu web graph, SK-2005 (S) is hyperlinks on '.sk' web. DelaunaySC (D) and Graph500 (G) are synthetic graphs. Notice that only (G) is an undirected graph.

Experimental setting: We evaluate our algorithm over all the graphs presented in Table 1, by varying the number of partitions K from 2 to 32. More precisely, we execute 10 runs of B-GRAP$_{\text{VB}}$ and B-GRAP$_{\text{EB}}$ for each graph and each value of K to ensure the significance of the results. For all experiments, we compute the average variation of the following measures with respect to the number of partitions K and over the runs:

– The maximum normalized unbalance of vertices ($\mathbf{MNU}_{\text{VB}}$) and of edges ($\mathbf{MNU}_{\text{EB}}$).
– The divergence between the distribution of vertices (respectively of edges) and the uniform distribution $\mathbf{JSD}_{\text{VB}}$ (respectively $\mathbf{JSD}_{\text{EB}}$).
– The edge-cuts ratio (**EC**).
– The computation time saving ratio ($\mathbf{\Delta Time}$) of B-GRAP w.r.t Spinner[1]. This ratio is computed using the total CPU time in seconds spent to execute the algorithm, from the initialization until the convergence.

[1] $\mathbf{\Delta Time} = \frac{Time(Spinner) - Time(\text{B-GRAP})}{Time(Spinner)}$.

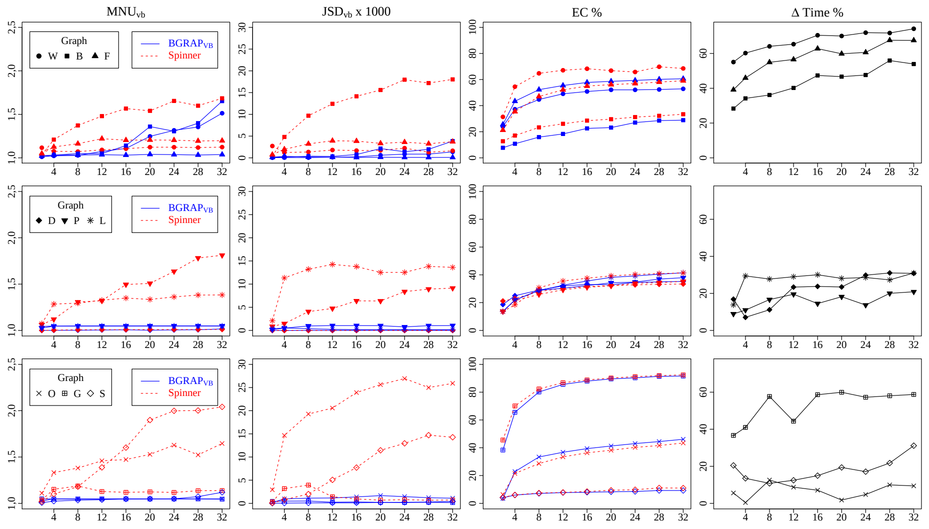

Fig. 2: Variation of the average scores of $\mathbf{MNU}_{\mathrm{VB}}$, $\mathbf{JSD}_{\mathrm{VB}}$, $\mathbf{EC}$, and $\mathit{\Delta}\mathbf{Time}$ for the partitioning obtained with B-GRAP$_{\mathrm{VB}}$ and Spinner, w.r.t. K

Note that $\mathit{\Delta}\mathbf{Time} > 0$ means a better performance of our algorithm and a value close to 0 means similar performances with Spinner.

For all experiments, we set $\epsilon = 10^{-3}$ as a threshold stop value and we set τ average out degree $\bar{d}^{+} = \frac{|E|}{|V|}$. The penalty term weight parameter λ in the update function $\mathcal{F}$ is set to 1. This gives an equal importance to the penalty term $\mathcal{P}$ and to $\mathcal{F}_{\mathrm{LP}}$ according to the update functions defined in Section 4.2.

6.2 B-GRAP Vertex balance:

The experiments presented in this section consider the vertex balance constraints to partition a graph. The main objective is to evaluate the ability of our algorithm B-GRAP$_{\mathrm{VB}}$ to produce balanced partitions with respect to the number of vertices, while improving the quality of cuts, using the vertex-balance update function defined in Eq. 6.

For this aim, using the experimental protocol described in Section 6.1, we compare the balance and the cuts quality of B-GRAP$_{\mathrm{VB}}$ partitioning with Spinner partitioning.

Results: The results are presented in Figure 2. This figure shows, for each graph and algorithm, the average variation of the $\mathbf{MNU}_{\mathrm{VB}}$, $\mathbf{JSD}_{\mathrm{VB}}$, $\mathbf{EC}$, and $\mathit{\Delta}\mathbf{Time}$ according the number of partitions K.

We analyze first the variation of the unbalance degree $\mathbf{MNU}_{\mathrm{VB}}$ and the total balance of the partitioning $\mathbf{JSD}_{\mathrm{VB}}$. As shown on the Figure 2, B-GRAP$_{\mathrm{VB}}$ produces generally a low unbalance degree for the most part of graphs (seven over nine w.r.t. $\mathbf{MNU}_{\mathrm{VB}}$) while varying the number of partitions K. On the other side, the results of Spinner show a high unbalance degree $\mathbf{MNU}_{\mathrm{VB}}$ (> 1.1) when scaling with K, in particular for $K \geq 4$, except for (D) graph. We notice only two exceptions for B-GRAP$_{\mathrm{VB}}$ on (B) and (W) graphs, when $K \geq 24$. However, the $\mathbf{MNU}_{\mathrm{VB}}$ of B-GRAP$_{\mathrm{VB}}$ is still lower then Spinner for (B) graph.

Similarly, the results of $\mathbf{JSD}_{\mathrm{VB}}$ show that B-GRAP$_{\mathrm{VB}}$ performs generally better than Spinner. The value of $\mathbf{JSD}_{\mathrm{VB}}$ is very close to 0 over all graphs and for all K. This means that B-GRAP$_{\mathrm{VB}}$ produces high balanced partitions. B-GRAP$_{\mathrm{VB}}$ gives better results for 6 graphs over 9 (with 5 significant differences for (B), (D), (P), (O), and (S) and similar results for the others). We note that for the exceptions on (W) and (B) noticed previously for $\mathbf{MNU}_{\mathrm{VB}}$, the $\mathbf{JSD}_{\mathrm{VB}}$ values are very close to 0 which means that the partitioning has a high global balance degree.

We compare the quality of cuts for both algorithms. Figure 2 shows similar quality of **EC**. B-GRAP$_{\mathrm{VB}}$ shows significant better results on *BerkeleyStanf* and *WikiTalk* graphs.

Finally, the $\mathbf{\Delta Time}$ curves show that B-GRAP$_{\mathrm{VB}}$ improves significantly the computation time on all graphs. The time saving percent $\mathbf{\Delta Time}$ is higher than 10% for all the graphs and all K values, except of (O) graph, where the the results are better but less significant.

6.3 B-GRAP Edge balance:

Now we compare the performance of our algorithm B-GRAP$_{\mathrm{EB}}$ with Spinner, using the edge-balance update function defined in Eq. 8.

Results: We present the results of this experiment in Figure 3. For each graph and algorithm we show the average variation of the following measures w.r.t. K: $\mathbf{MNU}_{\mathrm{EB}}$, $\mathbf{JSD}_{\mathrm{EB}}$, **EC**, and $\mathbf{\Delta Time}$.

Figure 3 shows that the partitioning produced by B-GRAP$_{\mathrm{EB}}$ has a low edge unbalance degree for all graphs under analysis. In fact, the average $\mathbf{MNU}_{\mathrm{EB}}$ is generally less than 1.05, except in the case of *WikiTalk* for $K = 28$ and $K = 32$ where the average $\mathbf{MNU}_{\mathrm{EB}}$ is equal to 1.12 and 1.13, respectively. However, if we analyze the results obtained from running Spinner, we see that we obtain an unbalance degree $\mathbf{MNU}_{\mathrm{EB}}$ generally higher than 1.05 and $\mathbf{MNU}_{\mathrm{EB}}$ shows bad values while increasing the number of partitions K. On the contrary, the variation of $\mathbf{MNU}_{\mathrm{EB}}$ for B-GRAP$_{\mathrm{EB}}$ shows that it scales with K with a stable balance quality.

The behaviour of $\mathbf{JSD}_{\mathrm{EB}}$ shows that B-GRAP$_{\mathrm{EB}}$ generally scales up with K while maintaining a good global balance, with few exceptions. Furthermore, B-GRAP$_{\mathrm{EB}}$ obtains better performance than Spinner over five graphs and gives similar $\mathbf{JSD}_{\mathrm{EB}}$ scores for the others.

The quality of cuts is generally close for both algorithms (Figure 3), with only one significant better result on *WikiTalk*.

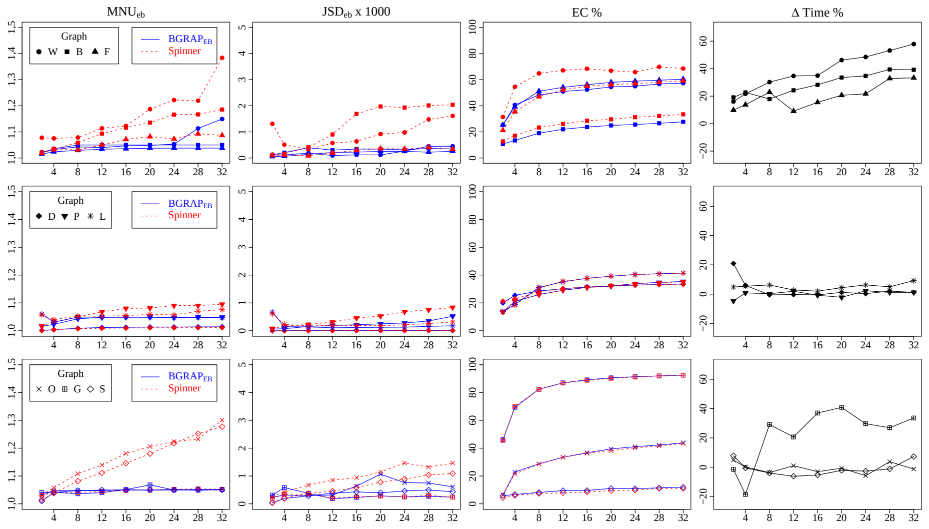

Fig. 3: Variation of the average scores of $\mathbf{MNU}_{\mathrm{EB}}$, $\mathbf{JSD}_{\mathrm{EB}}$, **EC** and $\mathit{\Delta}$**Time** for the partitioning obtained with B-GRAP$_{\mathrm{EB}}$ and Spinner, w.r.t. K

Finally, $\mathit{\Delta}$**Time** variation shows significant better results for B-GRAP$_{\mathrm{EB}}$ on (W), (B), (F) and (G) graphs. The computation time is slightly better for other graphs. In fact, this is only with few exceptions on (G) for $K \leq 4$ and for (S).

Summary: B-GRAP$_{\mathrm{EB}}$ algorithm computes higher edge balanced partitioning without impacting the quality of cuts and while showing generally faster computation time. Moreover, the results on the undirected graph *Graph500* show that the initialization step is efficient for the time execution of the algorithm. Finally, the results given for the $\mathbf{JSD}_{\mathrm{EB}}$ and $\mathbf{MNU}_{\mathrm{EB}}$ show that a better balance can be obtained if we consider the directness of edges for a directed graph.

7 Conclusion and Perspectives

In this paper we proposed two scalable and parallel partitioning algorithms B-GRAP$_{\mathrm{VB}}$ and B-GRAP$_{\mathrm{EB}}$, based on LP, that address the vertex balance and the edge balance problems respectively on both directed and undirected graphs. We defined the initialization strategy of our algorithm that allows to speed up the convergence and two update functions to produce either vertex balanced or edge balanced partitioning.

Our results show good performances of B-GRAP on various graphs and with different scales. We show that B-GRAP produces high vertex balanced and high edge balanced partitioning with a good cuts quality comparing to Spinner algorithm (significant values for 5 graphs and slightly better values for others), on either directed and undirected graphs. Moreover, the computation time of

B-GRAP is better than Spinner, with few exception for B-GRAP$_{\mathrm{EB}}$ on two graphs.

The additional experiments we conducted to study the initialization step B-GRAP$_{init}$ show that the selection of the seed vertices has an impact on the quality of the partitioning and the computation time. We would study more deeply this step in order to optimize our method.

We would also study the impact of the partitioning on algorithms of graph analytics with respect to the balance strategy, such as Shortest Path Computation, PageRank, and Community Detection.

References

1. Backstrom, L., Huttenlocher, D., Kleinberg, J., Lan, X.: Group formation in large social networks. In: Proc. of the 12th ACM SIGKDD Inter. Conf. on Knowledge discovery and data mining. p. 44. ACM Press (2006)
2. Bader, D., Meyerhenke, H., Wagner, D.: Graph Partitioning and Graph Clustering, Contemporary Mathematics, vol. 588 (2013)
3. Buluç, A., Meyerhenke, H., Safro, I., Sanders, P., Schulz, C.: Recent advances in graph partitioning. In: Lect. Notes Comput. Sci., vol. 9220, pp. 117–158 (2016)
4. Chakraborty, T., Dalmia, A., Mukherjee, A., Ganguly, N.: Metrics for community analysis: A survey **50**(4), 1–37 (2016)
5. Chevalier, C., Pellegrini, F.: PT-Scotch: A tool for efficient parallel graph ordering. Tech. Rep. 6-8 (2008)
6. Das, H., Kumar, S.: A parallel TSP-based algorithm for balanced graph partitioning. In: 2017 46th Inter. Conf. on Parallel Processing (ICPP). pp. 563–570. IEEE (2017)
7. Giraph, A.: Giraph : Large-scale graph processing in Hadoop (2012)
8. Gonzalez, J.E., Low, Y., Gu, H., Bickson, D., Guestrin, C.: PowerGraph: Distributed graph-parallel computation on natural graphs. In: Proc. of the 10th USENIX Conf. on Operating Systems Design and Implementation. pp. 17–30 (2012)
9. Gregory, S.: Finding overlapping communities in networks by label propagation **12**(10), 103018 (2010)
10. Heidari, S., Simmhan, Y., N. Calheiros, R., Buyya, R.: Scalable graph processing frameworks: A taxonomy and open challenges **51**, 1–53 (2018)
11. Henzinger, A., Noe, A., Schulz, C.: ILP-based local search for graph partitioning (2018)
12. Karypis, G., Kumar, V.: Multilevel graph partitioning schemes. In: Proc. of the 24th Inter. Conf. on Parallel Processing (ICPP) 1955. vol. 3, pp. 113–122 (1995)
13. Karypis, G., Kumar, V.: A parallel algorithm for multilevel graph partitioning and sparse matrix ordering **48**(1), 71–95 (1998)
14. Leskovec, J., Huttenlocher, D., Kleinberg, J.: Signed networks in social media. In: Proc. of the 28th Inter. Conf. on Human factors in computing systems. p. 1361 (2010)
15. Leskovec, J., Krevl, A.: SNAP datasets: Stanford large network dataset collection (2014), https://snap.stanford.edu/data/index.html
16. Leskovec, J., Lang, K.J., Dasgupta, A., Mahoney, M.W.: Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters **6**(1), 29–123 (2009)

17. Li, Y., Constantin, C., du Mouza, C.: A Block-Based Edge Partitioning for Random Walks Algorithms over Large Social Graphs. In: Proc. 17th Int. Conf. Web Inf. Syst. Eng. (WISE)- Vol. 10042, pp. 275–289. Springer-Verlag New York, Inc. (2016)
18. Lin, J.: Divergence measures based on the shannon entropy. IEEE Transactions on Information Theory **37**(1), 145–151 (Jan 1991)
19. Malewicz, G., Austern, M.H., Bik, A.J., Dehnert, J.C., Horn, I., Leiser, N., Czajkowski, G.: Pregel: A system for large-scale graph processing. In: Proc. of the 2010 ACM SIGMOD Inter. Conf. on Management of Data. pp. 135–146 (2010)
20. Martella, C., Logothetis, D., Loukas, A., Siganos, G.: Spinner: Scalable graph partitioning in the cloud. In: Proc. - Int. Conf. Data Engineering (2017)
21. Meyerhenke, H., Sanders, P., Schulz, C.: Parallel graph partitioning for complex networks. vol. 28, pp. 2625–2638 (2017)
22. Nguyen, D.: Graph Partitioning. ISTE (2011)
23. Pellegrini, F., Roman, J.: Scotch: A software package for static mapping by dual recursive bipartitioning of process and architecture graphs. In: Proc. of the Inter. Conf. and Exhibition on High-Performance Computing and Networking. pp. 493–498. HPCN Europe 1996 (1996)
24. Raghavan, U.N., Albert, R., Kumara, S.: Near linear time algorithm to detect community structures in large-scale networks. p. 036106 (2007)
25. Rossi, R.A., Ahmed, N.K.: The network data repository with interactive graph analytics and visualization. In: Proc. of the 29 AAAI (2015)
26. Sanders, P., Schulz, C.: Think Locally, Act Globally: Highly Balanced Graph Partitioning. In: Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13). LNCS, vol. 7933, pp. 164–175. Springer (2013)
27. Tsourakakis, C., Gkantsidis, C., Radunovic, B., Vojnovic, M.: FENNEL: Streaming graph partitioning for massive scale graphs. In: Proc. of the 7th ACM Inter. Conf. on Web search and data mining. pp. 333–342 (2014)
28. Ugander, J., Backstrom, L.: Balanced label propagation for partitioning massive graphs. In: Proc. of the 6th ACM Inter. Conf. on Web Search and Data Mining. pp. 507–516. WSDM '13, ACM (2013)
29. Xin, R.S., Gonzalez, J.E., Franklin, M.J., Stoica, I.: GraphX: a resilient distributed graph system on spark. In: First International Workshop on Graph Data Management Experiences and Systems. pp. 1–6. ACM Press (2013)
30. Zafarani, R., Liu, H.: Users joining multiple sites: Distributions and patterns (2014)