Distributed Power-law Graph Computing: Theoretical and Empirical Analysis

Cong Xie xcgoner1108@gmail.com
Department of Computer Science and Engineering
Shanghai Jiao Tong University
800 Dong Chuan Road, Shanghai, China 200240

Ling Yan yling0718@sjtu.edu.cn
Department of Computer Science and Engineering
Shanghai Jiao Tong University
800 Dong Chuan Road, Shanghai, China 200240

Wu-Jun Li liwujun@nju.edu.cn
National Key Laboratory for Novel Software Technology
Collaborative Innovation Center of Novel Software Technology and Industrialization
Department of Computer Science and Technology, Nanjing University
Nanjing, Jiangsu, China, 210023

Zhihua Zhang zhang-zh@cs.sjtu.edu.cn
Department of Computer Science and Engineering
Shanghai Jiao Tong University
800 Dong Chuan Road, Shanghai, China 200240

Editor: Vahab Mirrokni

Abstract

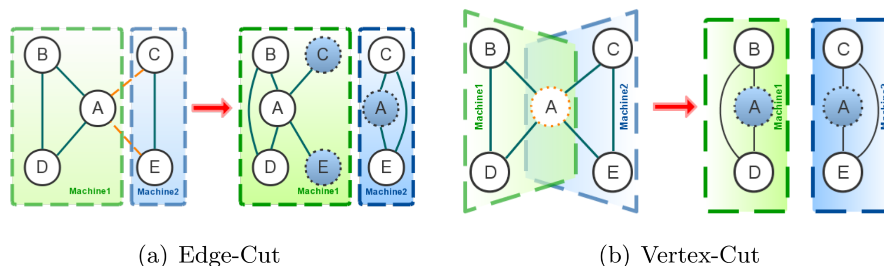
With the emergence of big graphs in a variety of real applications like social networks, machine learning based on distributed graph-computing (DGC) frameworks has attracted much attention from large-scale machine learning community. In the DGC framework, the graph partitioning (GP) strategy plays a key role to affect the performance, including the communication cost and workload balance. Typically, the degree distributions of natural graphs from real applications follow power laws, which makes GP a challenging task. Recently, many methods have been proposed to solve the GP problem. However, the existing GP methods cannot achieve satisfactory performance for applications with power-law graphs. In this paper, we propose a novel vertex-cut GP framework, called PowerLore, by making effective use of the power-law degree distribution in graphs. PowerLore includes a degree-based hashing algorithm (called Libra) and two degree-based greedy algorithms (called Constell and Zodiac). Theoretical analysis shows that Libra can achieve lower communication cost than existing hashing-based methods and can simultaneously guarantee good workload balance. Empirical results show that the two degree-based greedy algorithms can further improve the performance of Libra which is non-greedy. Moreover, empirical results on several large power-law graphs also show that PowerLore can outperform existing state-of-the-art GP methods.

Keywords: Distributed Graph Computing, Graph Partitioning, Power Law, Large-scale Machine Learning

1. Introduction

Recent years have witnessed the emergence of big graphs in a large variety of real applications, such as the web and social network services. Furthermore, many machine learning and data mining algorithms can also be modeled with graphs (Low et al., 2010). Hence, machine learning based on distributed graph-computing (DGC) frameworks has attracted much attention from large-scale machine learning community (Low et al., 2010; Malewicz et al., 2010; Low et al., 2012; Gonzalez et al., 2012; Kyrola et al., 2012; Gonzalez et al., 2014). To perform distributed graph-computing on clusters with several machines (servers), one has to partition the whole graph across the machines in a cluster. Graph partitioning (GP) can dramatically affect the performance of DGC frameworks in terms of communication cost and workload balance. Therefore, the GP strategy typically plays a key role in DGC frameworks. The ideal GP method should minimize the cross-machine communication cost, and simultaneously keep the workload in every machine approximately balanced.

Existing GP methods can be divided into two main categories: edge-cut and vertex-cut methods. Edge-cut tries to evenly assign the vertices to machines by cutting the edges. In contrast, vertex-cut tries to evenly assign the edges to machines by cutting the vertices. Figure 1 illustrates the edge-cut and vertex-cut partitioning results of an example graph. In Figure 1(a), the edges (A,C) and (A,E) are cut, and the two machines store the vertex sets {A,B,D} and {C,E}, respectively. In Figure 1(b), the vertex A is cut, and the two machines store the edge sets {(A,B), (A,D), (B,D)} and {(A,C), (A,E), (C,E)}, respectively. In edge-cut, both machines of a cut edge should maintain a ghost (local replica) of the vertex and the edge data. In vertex-cut, all the machines associated with a cut vertex should maintain a mirror (local replica) of the vertex. The ghosts and mirrors are shown in shaded vertices in Figure 1. In edge-cut, the workload of a machine is determined by the number of vertices located in that machine, and the communication cost of the whole graph is determined by the number of edges spanning different machines. In vertex-cut, the workload of a machine is determined by the number of edges located in that machine, and the communication cost of the whole graph is determined by the number of mirrors of the vertices.

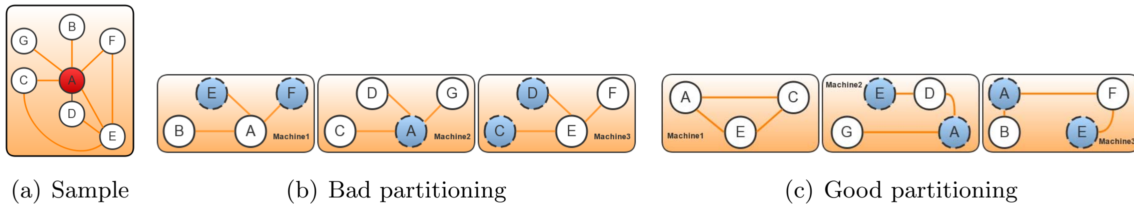


(a) Edge-Cut (b) Vertex-Cut

Figure 1: Two strategies for graph partitioning. Shaded vertices are ghosts and mirrors respectively. (a) The edges (A,C) and (A,E) are cut, and the two machines store respectively the vertex sets {A,B,D} and {C, E} with all the corresponding edges. (b) The vertex A is cut, and the two machines store respectively the edge sets {(A,B), (A,D), (B,D)} and {(A,C), (A,E), (C,E)} with all the corresponding vertices.

Most traditional DGC frameworks, such as GraphLab (Low et al., 2010) and Pregel (Malewicz et al., 2010), use edge-cut methods (Karypis and Kumar, 1995; Stanton and Kliot, 2012; T-sourakakis et al., 2014; Wang et al., 2014) for GP. However, the authors of PowerGraph (Gonzalez et al., 2012) found that vertex-cut methods can achieve better performance than edge-cut methods, especially for power-law graphs. Therefore, vertex-cut has attracted more and more attention from DGC research community. For example, PowerGraph (Gonzalez et al., 2012) adopts a *random* vertex-cut method and two greedy variants for GP. GraphBuilder (Jain et al., 2013) provides some heuristics, such as the *grid*-based constrained solution, to improve the random vertex-cut method.

The degree distributions of large natural graphs usually follow power laws, which makes GP challenging. Different vertex-cut methods can result in different performance for power-law graphs. For example, Figure 2(a) shows a toy power-law graph with only one vertex having much higher degree than the others. Figure 2(b) shows a partitioning strategy by cutting the vertices {E, F, A, C, D}, and Figure 2(c) shows a partitioning strategy by cutting the vertices {A, E}. We can see that the partitioning strategy in Figure 2(c) is better than that in Figure 2(b) because the number of mirrors (shaded vertices) in Figure 2(c) is smaller which means less communication cost. The intuition underlying this example is that cutting higher-degree vertices can result in fewer mirror vertices. Therefore, the power-law degree distribution can be used to facilitate GP.



(a) Sample (b) Bad partitioning (c) Good partitioning

Figure 2: Partition a sample graph with vertex-cut.

Unfortunately, existing vertex-cut methods, including those in PowerGraph and GraphBuilder, make rare use of the power-law degree distribution for GP. Thus, they cannot achieve satisfactory performance in natural power-law graphs. Very recently, PowerLyra (Chen et al., 2015) is proposed to combine both edge-cut and vertex-cut together by using the power-law degree distribution. However, it lacks theoretical guarantee.

In this paper, we propose a novel vertex-cut GP framework, called PowerLore, for distributed power-law graph computing. The main contributions of PowerLore are briefly outlined as follows:

- In PowerLore, a degree-based hashing algorithm (called Libra) and two degree-based greedy algorithms (called Constell and Zodiac) are proposed for GP. These algorithms can make effective use of the power-law degree distribution in natural graphs for vertex-cut GP.

- Theoretical bounds on the communication cost and workload balance for Libra can be derived, which show that Libra can achieve lower communication cost than existing hashing-based methods and simultaneously guarantee good workload balance.

- Compared with Libra which is non-greedy, the two degree-based greedy algorithms (Constell and Zodiac) can further improve the performance.

- PowerLore can be implemented as an execution engine for PowerGraph (Gonzalez et al., 2012), and all PowerGraph applications can be seamlessly supported by PowerLore.

- Empirical results on several large real graphs and synthetic graphs show that PowerLore can outperform existing state-of-the-art methods.

The remainder of this paper is organized as follows. Section 2 introduces the problem definition of this paper. In Section 3 we present the degree-based hashing algorithm and establish theoretical analysis. In Section 4 we present the two degree-based greedy algorithms. In Section 5 we conduct empirical evaluations. Finally, we conclude the paper in Section 6.

2. Problem Definition

In this section, we first introduce the vertex-programming model which is used by both PowerGraph and our PowerLore framework. Then the objective of vertex-cut GP is presented. Finally, the issues associated with GP for power-law graphs are introduced.

2.1 Vertex-Programming Model of PowerGraph

Our PowerLore framework uses the same vertex-programming model as that of PowerGraph, which is called the Gather-Apply-Scatter (GAS) vertex-programming model (Gonzalez et al., 2012). The GAS model decomposes a graph computing program into three phases: gather, apply, and scatter. Such abstraction naturally retains the "think-like-a-vertex" characteristic of Pregel (Malewicz et al., 2010). For each vertex, the user can define the three phases (gather, apply, scatter) of a GAS vertex-program. Once a DGC task is started, each vertex will start the user-defined vertex-program simultaneously. All the vertex-programs run without conflicts and update the values of the vertices and edges until the user-defined stopping criterion is satisfied.

A vertex-program needs to read the values of the central vertex v, all the adjacent edges (v, u) and all the adjacent vertices u. Each iteration of the GAS model is defined as follows: in the *gather* phase, the values of the adjacent vertices and edges are collected and then summed via a user-defined commutative and associative sum operation; in the *apply* phase, the final summed value resulted from the gather phase is used to update the value of the central vertex v; finally, in the *scatter* phase, the new value of the central vertex is used to update the values of the adjacent edges. A detailed interface of the GAS model is shown in Figure 3, where the corresponding values of v, (v, u) and u are denoted by $val(v)$, $val(v, u)$, and $val(u)$, respectively. The user should implement the *gather*, *sum*, *apply*, and *scatter* functions in the interface. A simple example of the GAS vertex-program for PageRank is shown in Figure 4, where outNbrs(u) denotes the out-degree of the vertex u.

```
GASVertexProgram(v) {
        // Run on central vertex v
        // Accumulators is a container
        Container accumulators;
        // Run in parallel for all the adjacent vertices and
           edges
        accumulators = gather(v, u);
        accumulator = sum(accumulators);
        val(v) = apply(v, accumulator);
        // Run in parallel for all the adjacent edges
        val(v, u) = scatter(v, u);
}
```

Figure 3: GAS vertex-program

```
PageRank(v) {
        gather(v, u) {
                return val(u).rank / outNbrs(u);
        }
        sum(a, b) {
                return a + b;
        }
        apply(v, accumulator) {
                new_rank = 0.85 * accumulator + 0.15;
                val(v).delta = (new_rank - val(v).rank) /
                   outNbrs(v);
                val(v).rank = new_rank;
                return val(v);
        }
        scatter(v, u) {
                if(abs(val(v).delta) > epsilon)
                        Activate(u);
                return val(v).delta;
        }
}
```

Figure 4: GAS vertex-program for PageRank

2.2 Vertex-Cut GP

Assume we have a cluster of p machines. Vertex-cut GP is to assign each edge with the two corresponding vertices to one of the p machines in the cluster. The assignment of an edge is unique, while vertices may have replicas across different machines (partitions). It is the synchronization operation over all the replicas of the same vertex that incurs communica-

tion. If a certain vertex has more than one replica, one of them will be set to be the *master* and all the others will be *mirrors*. To synchronize the values, the mirrors will send the local values to the master. Note that in the GAS vertex-programming model, the *gather* operation is commutative and associative. Therefore, all the local values associated with the same vertex can be summed up locally before sending to the master for synchronization. Finally, the master sends the summed value back and the synchronization is completed. So the communication cost for each vertex is linear to the number of mirrors. Since the number of replicas is always the number of mirrors plus one, the communication cost is also linear to the number of replicas. The workload (computation cost) is mainly from the *gather* phase. More specifically, the computation cost is mainly from the summation over the collected values of the adjacent vertices. So, the workload of a certain vertex is approximately linear to its degree. Furthermore, the workload of a certain partition (machine) will be approximately linear to the number of edges assigned to it. There is also some computation cost from the *apply* phase as well. Such cost (workload) of a machine is roughly linear to the number of master vertices on that machine.

Let $G = (V, E)$ denote a graph, where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of vertices and $E \subseteq V \times V$ is the set of edges in G. Let $|V|$ denote the cardinality of the set V, i.e., $n = |V|$. Vertices v_i and v_j are called neighbors if $(v_i, v_j) \in E$. We denote the set of all the neighbors of vertex v_i as $N(v_i) = \{v_j \in V \mid (v_i, v_j) \in E\}$, and the degree of vertex v_i is denoted as $d_i = |N(v_i)|$. Note that we only need to consider the GP task for undirected graphs because the communication cost and workload mainly depend on the number of edges for either directed or undirected graphs. Even if the computation is based on directed graphs, we can also use the undirected counterparts of the directed graphs to get the partitioning results.

Let $M(e) \in [p]$ be the machine that edge $e \in E$ is assigned to, and $A(v) \subseteq [p]$ be the span of vertex v over different machines. We can see that $|A(v)|$ is the number of replicas of v among different machines. Here we denote $[p] = \{1, \ldots, p\}$ for the sake of convenience. Similar to PowerGraph (Gonzalez et al., 2012), one of the replicas of a vertex is chosen as the *master* and the others are treated as the *mirrors* of the master. We let $Master(v)$ denote the machine in which the master of v is located.

As stated above, for DGC frameworks based on vertex-cut GP, the communication cost for synchronization is roughly linear to the number of replicas of the vertices, and the workload of a machine is roughly linear to the number of edges located in that machine. So the goal of vertex-cut GP is to *minimize* the number of replicas and simultaneously *balance* the number of edges and number of master nodes on each machine, which can be formulated as follows:

$$\min_{A} \frac{1}{n} \sum_{i=1}^{n} |A(v_i)| \tag{1}$$

$$s.t. \ \max_{m} |\{e \in E \mid M(e) = m\}| < \lambda \frac{|E|}{p} \text{ , and } \ \max_{m} |\{v \in V \mid Master(v) = m\}| < \rho \frac{n}{p},$$

where $m \in [p]$ denotes a machine, $\lambda \geq 1$ and $\rho \geq 1$ are imbalance factors. We define $\frac{1}{n}\sum_{i=1}^{n} |A(v_i)|$ as *replication factor*, $\frac{p}{|E|} \max_{m} |\{e \in E \mid M(e) = m\}|$ as *edge-imbalance*, and $\frac{p}{n} \max_{m} |\{v \in V \mid Master(v) = m\}|$ as *vertex-imbalance*. To get a good (high) *workload*

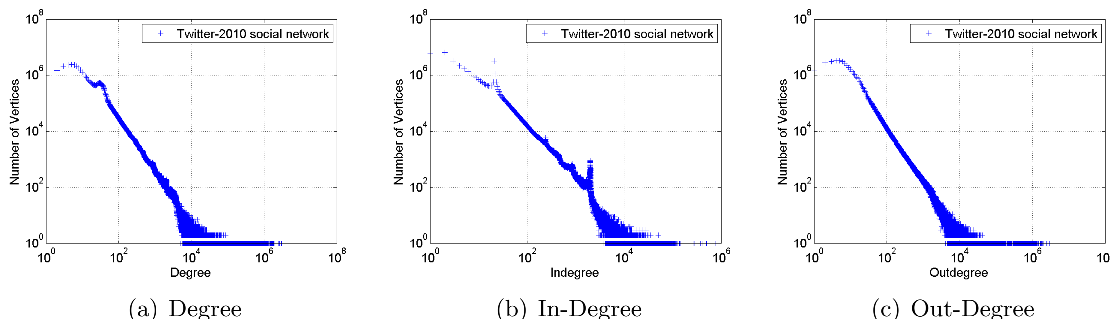
balance, λ and ρ should be as small as possible, which also means that the *edge-imbalance* and *vertex-imbalance* should be as small as possible.

2.3 GP for Power-Law Graphs

Typically, the degree distributions of natural graphs follow power laws (Adamic and Huberman, 2002):

$$\Pr(d) \propto d^{-\alpha},$$

where $\Pr(d)$ is the probability that a vertex has degree d and the power parameter $\alpha > 0$ is a positive constant. For example, $\alpha \approx 2$ for all the degree distribution, in-degree distribution and out-degree distribution in the Twitter-2010 social network (Kwak et al., 2010), which is shown in Figure 5. This power-law degree distribution makes GP challenging (Gonzalez et al., 2012). Although vertex-cut methods can achieve better performance than edge-cut methods for power-law graphs (Gonzalez et al., 2012), existing vertex-cut methods, such as *random* method in PowerGraph and *grid*-based method in GraphBuilder (Jain et al., 2013), cannot make effective use of the power-law distribution to achieve satisfactory performance.



(a) Degree (b) In-Degree (c) Out-Degree

Figure 5: Degree distribution of Twitter-2010 social network.

As observed in Figure 2, the power-law degree distribution might be used to facilitate GP. This motivates the work in this paper, which tries to exploit the power-law degree distribution to get a new GP framework called *PowerLore*. PowerLore includes a degree-based hashing algorithm (Libra) and two degree-based greedy algorithms (Constell and Zodiac), which will be introduced in the following two sections.

3. Degree-based Hashing Algorithm

In this section, we present a degree-base hashing algorithm to effectively exploit the power-law degree distribution for vertex-cut GP. In addition, theoretical analysis is also established.

3.1 Hashing Model

We refer to a certain machine by its index idx, and the idxth machine is denoted as P_{idx}. First, we define two kinds of hash functions:

$$idx = vertex_hash(v_i) \in [p],$$

which hashes some vertex v_i to the machine P_{idx}, and

$$idx = edge_hash(e) \in [p]$$

or

$$idx = edge_hash(v_i, v_j) \in [p],$$

which hashes some edge $e = (v_i, v_j)$ to the machine P_{idx}.

Our *hashing model* includes two main components:

- *Master-vertex assignment*: The *master* replica of v_i is uniquely assigned to one of the p machines with equal probabilities by some *random* hash function $vertex_hash(v_i)$.
- *Edge assignment*: Each edge $e = (v_i, v_j)$ is uniquely assigned to one of the p machines by some hash function $edge_hash(v_i, v_j)$.

Please note that $vertex_hash(\cdot)$ is a *random* function, but $edge_hash(\cdot)$ is a multi-to-one mapping which will be defined in the next subsection. It is easy to check that the above hashing model is a vertex-cut GP method. The *master-vertex assignment* can be easily implemented, which can also be expected to achieve a low vertex-imbalance score. On the contrary, the *edge assignment* is much more complicated. Different edge-hash functions can achieve different *replication factors* and different *edge-imbalance* scores. Note that replication factor reflects communication cost, and edge-imbalance reflects workload-imbalance. Thus, the key of our hashing model lies in the edge-hash function $edge_hash(\cdot)$.

3.2 Degree-based Hashing Algorithm

From the example in Figure 2, we observe that the replication factor, which is defined as the total number of replicas divided by the total number of vertices, will be smaller if we cut more vertices with relatively higher degrees. Based on this intuition, we define the $edge_hash(v_i, v_j)$ as follows:

$$edge_hash(v_i, v_j) = \begin{cases} vertex_hash(v_i) & \text{if } d_i < d_j, \\ vertex_hash(v_j) & \text{otherwise.} \end{cases}$$

It means that we use the vertex-hash function to define the edge-hash function. Furthermore, the edge-hash function value of an edge is determined by the degrees of the two associated vertices. More specifically, the edge-hash function value of an edge is defined by the vertex-hash function value of the associated vertex with a smaller degree. Therefore, our method is a degree-based hashing algorithm. We name this algorithm by *Libra*, because it weighs the degrees of the two ends of each edge, which behaves like a pair of scales. Libra can effectively capture the intuition that cutting vertices with higher degrees will get better performance.

The degree-based hashing algorithm (Libra) for vertex-cut GP is briefly summarized in Algorithm 1.

From Algorithm 1, we can find that for a certain vertex v_i, all its adjacent edges $N_e(v_i) = \{e = (v_i, v_j) \in E \mid v_j \in N(v_i)\}$ are separated into two parts:

Algorithm 1 Degree-based Hashing Algorithm (Libra)

Input: The set of edges E; the set of vertices V; the number of machines p.
Output: The assignment $M(e) \in [p]$ of each edge e.

1: Count the degree d_i of each $i \in [n]$ in parallel ▷ Initialization
2: **for all** $e = (v_i, v_j) \in E$ **do** ▷ Partitioning
3: Hash each edge in parallel
4: **if** $d_i < d_j$ **then**
5: $M(e) \leftarrow vertex_hash(v_i)$
6: **else**
7: $M(e) \leftarrow vertex_hash(v_j)$
8: **end if**
9: **end for**

- The edges hashed by v_i (i.e., $edge_hash(v_i, v_j) = vertex_hash(v_i)$). We refer to this case as *centrally hashed*.
- The edges hashed by $v_j \in N(v_i)$ (i.e., $edge_hash(v_i, v_j) = vertex_hash(v_j)$). We refer to this case as *decentrally hashed*.

3.3 Theoretical Analysis

In this subsection, we present theoretical analysis for our *Libra* algorithm. For comparison, the *random* vertex-cut method (called *Random*) of PowerGraph (Gonzalez et al., 2012) and the *grid*-based constrained solution (called *Grid*) of GraphBuilder (Jain et al., 2013) are adopted as baselines. Our analysis is based on randomization. Moreover, we assume that the graph is undirected and there are no multiedges in the graph. We mainly study the performance in terms of *replication factor*, and *edge-imbalance* defined in Section 2.2. All the proofs can be found in the Appendices at the end of this paper.

3.3.1 Partitioning Degree-fixed Graphs

Firstly, we assume that the degree sequence $\{d_i\}_{i=1}^n$ is fixed. Then we can get the following expected replication factor produced by different methods.

Random assigns each edge evenly to p machines via a random hash function. The following result can be directly obtained from PowerGraph (Gonzalez et al., 2012).

Lemma 1 *Assume that we have a sequence of n vertices $\{v_i\}_{i=1}^n$ and the corresponding degree sequence $D = \{d_i\}_{i=1}^n$. A simple random vertex-cut on p machines has the expected replication factor:*

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}|A(v_i)|\,\middle|\,D\right] = \frac{p}{n}\sum_{i=1}^{n}\left[1 - \left(1 - \frac{1}{p}\right)^{d_i}\right].$$

Let $(p_1, p_2) = \operatorname{argmin}_{p_1,p_2} |p_1 - p_2|$, where $p_1, p_2 \in \mathbb{N}$ and $p_1 \times p_2 = p$. So, for $\sqrt{p} \in \mathbb{N}$, $p_1 = p_2 = \sqrt{p}$. When using the *Grid* hash function, each vertex has $p_1 + p_2 - 1$ rather than p candidate machines compared to the simple random hash function. Then we can derive the following corollary.

Corollary 2 *If Grid-hashing (Jain et al., 2013) is used, the expected replication factor on p machines is:*

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}|A(v_i)|\middle|D\right]$$
$$=\frac{1}{n}\sum_{i=1}^{n}\left\{(p_1-1)\left[1-\Big(1-a_1\Big)^{d_i}\right]+(p_2-1)\left[1-\Big(1-a_2\Big)^{d_i}\right]+\left[1-\Big(1-a_3\Big)^{d_i}\right]\right\},$$

where $a_1=\frac{p_2-1}{p}\times\frac{1}{2}+\frac{p_1-1}{p}\times\frac{1}{p_1}+\frac{1}{p}\times\frac{1}{p_1+p_2-1}$, $a_2=\frac{p_1-1}{p}\times\frac{1}{2}+\frac{p_2-1}{p}\times\frac{1}{p_2}+\frac{1}{p}\times\frac{1}{p_1+p_2-1}$, *and* $a_3=\frac{p_1-1}{p}\times\frac{1}{p_1}+\frac{p_2-1}{p}\times\frac{1}{p_2}+\frac{1}{p}\times\frac{1}{p_1+p_2-1}$.

Define $H=\{h_i\}_{i=1}^{n}$, where h_i denotes the number of v_i's adjacent edges which are decentrally hashed. It means that d_i-h_i adjacent edges are centrally hashed for v_i.

Theorem 3 *Assume that we have a sequence of n vertices* $\{v_i\}_{i=1}^{n}$ *and the corresponding degree sequence* $D=\{d_i\}_{i=1}^{n}$. *Our Libra algorithm on p machines has the expected replication factor:*

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}|A(v_i)|\middle|H,D\right]=\frac{p}{n}\sum_{i=1}^{n}\left[1-\Big(1-\frac{1}{p}\Big)^{h_i+g_i}\right]\le\frac{p}{n}\sum_{i=1}^{n}\left[1-\Big(1-\frac{1}{p}\Big)^{d_i}\right],$$

where

$$g_i=\begin{cases}1 & \text{if } h_i<d_i,\\ 0 & \text{if } h_i=d_i.\end{cases}$$

This theorem says that *Libra* has a smaller expected replication factor than *Random* of PowerGraph (Gonzalez et al., 2012).

Next we turn to the analysis of the balance constraints. We still fix the degree sequence. To simplify the analysis, we add some assumptions.

Assumption 1 *The partitioning result of random hash function vertex_hash(·) is ideally balanced. That is, there are approximately n/p master vertices in each partition.*

Assumption 2 $p\ge 2$ *and* $p\ll n$.

Assumption 3 *There is no self-loops and multiedges in the graph.*

Under such assumptions, *Random* has approximately perfect edge-balance. For *Libra*, we have the following desirable result.

Theorem 4 *Our Libra algorithm on p machines has the edge-imbalance:*

$$\frac{\max_m|\{e\in E\mid M(e)=m\}|}{|E|/p}\approx\frac{\sum_{i=1}^{n}\frac{h_i}{p}+\max_{j\in[p]}\sum_{v_i\in P_j}(d_i-h_i)}{2|E|/p}.$$

3.3.2 Partitioning Power-law Graphs

Now we change the sequence of fixed degrees into a sequence of random samples generated from the power-law distribution. As a result, upper-bounds can be provided for the above three methods: *Random*, *Grid* and *Libra*.

We add the following assumption to simplify the analysis.

Assumption 4 *$d_{min} \geq 1$, where d_{min} denotes the minimal degree in $D = \{d_i\}_{i=1}^n$.*

Theorem 5 *Assume that each $d \in \{d_i\}_{i=1}^n$ is sampled from a power-law degree distribution with parameter $\alpha > 0$. The expected replication factor of Random on p machines can be bounded by:*

$$\mathbb{E}_D\left[\frac{p}{n}\sum_{i=1}^n\left(1-\left(1-\frac{1}{p}\right)^{d_i}\right)\right] \leq p\left[1-\left(1-\frac{1}{p}\right)^{\hat{\Omega}}\right],$$

$$\text{where } \hat{\Omega} = \begin{cases} \frac{\alpha-1}{\alpha-2} \times \frac{d_{min}^{2-\alpha}-(n-1)^{2-\alpha}+(\alpha-2)d_{min}^{1-\alpha}}{d_{min}^{1-\alpha}-n^{1-\alpha}} & \text{if } \alpha \neq 1 \text{ and } \alpha \neq 2, \\ \frac{n-d_{min}}{\ln(n)-\ln(d_{min})} & \text{if } \alpha = 1, \\ \frac{\ln(n-1)-\ln(d_{min})+d_{min}^{-1}}{d_{min}^{-1}-n^{-1}} & \text{if } \alpha = 2. \end{cases}$$

Theorem 5 says that when the degree sequence follows the power-law distribution, the upper bound of the expected replication factor increases as α decreases.

Like Corollary 2, we get the similar result for *Grid*.

Corollary 6 *By using Grid method, the expected replication factor on p machines can be bounded by:*

$$\mathbb{E}_D\left[\frac{1}{n}\sum_{i=1}^n\left\{(p_1-1)\left[1-\left(1-a_1\right)^{d_i}\right]+(p_2-1)\left[1-\left(1-a_2\right)^{d_i}\right]+\left[1-\left(1-a_3\right)^{d_i}\right]\right\}\right]$$
$$\leq (p_1-1)\left[1-\left(1-a_1\right)^{\hat{\Omega}}\right]+(p_2-1)\left[1-\left(1-a_2\right)^{\hat{\Omega}}\right]+\left[1-\left(1-a_3\right)^{\hat{\Omega}}\right].$$

Note that $(p_1-1)\left[1-\left(1-a_1\right)^{\hat{\Omega}}\right]+(p_2-1)\left[1-\left(1-a_2\right)^{\hat{\Omega}}\right]+\left[1-\left(1-a_3\right)^{\hat{\Omega}}\right] \leq p\left[1-\left(1-\frac{1}{p}\right)^{\hat{\Omega}}\right]$. So Corollary 6 tells us that *Grid* can achieve smaller replication factor than Random.

Theorem 7 *Assume that each $d \in \{d_i\}_{i=1}^n$ is sampled from a power-law degree distribution with parameter $\alpha > 0$. When n is very large, the gap between the replication factors of Random and Libra increases expectedly as α decreases. The expected replication factor of Libra on p machines can be bounded by:*

$$\mathbb{E}_{H,D}\left[\frac{p}{n}\sum_{i=1}^n\left(1-\left(1-\frac{1}{p}\right)^{h_i+g_i}\right)\right] \leq p\left[1-\left(1-\frac{1}{p}\right)^{\hat{\Omega}/2+\mathbb{E}[g_i]}\right] < p\left[1-\left(1-\frac{1}{p}\right)^{\hat{\Omega}/2+1}\right].$$

Note that

$$p\left[1-\left(1-\frac{1}{p}\right)^{\hat{\Omega}/2+1}\right] \le p\left[1-\left(1-\frac{1}{p}\right)^{\hat{\Omega}}\right],$$

when $\hat{\Omega} \ge 2$. Thus, Libra can expectedly reduce the replication factor compared to *Random*.

Note that *Grid* and our degree-based hashing algorithm Libra actually use two distinct ways to reduce the replication factor. *Grid* reduces more replication factor when p grows, which is independent of the degree distribution. In contrast, Libra proposes to exploit the power-law degree distribution. It is not important to show which one reduces more replicas. Actually, these two approaches can be easily combined to obtain further improvement.

Finally, we show that Libra also guarantees good edge-balance.

Theorem 8 *Assume each edge is hashed by Libra with* $\{v_i\}_{i=1}^n$, $\{d_i\}_{i=1}^n$ *and* $\{h_i\}_{i=1}^n$ *defined above, and the vertices are evenly assigned. By taking the constant* $2|E|/p = \mathbb{E}_D\left[\sum_{i=1}^n d_i\right]/p = n\mathbb{E}_D[d]/p$, *there exists a small positive constant* ϵ *such that the expected edge-imbalance of Libra on* p *machines can be bounded w.h.p (with high probability). That is,*

$$\mathbb{E}_{H,D}\left[\sum_{i=1}^n \frac{h_i}{p} + \max_{j\in[p]} \sum_{v_i\in P_j} (d_i - h_i)\right] \le (1+\epsilon)\frac{2|E|}{p}.$$

Therefore, with Theorem 7 together, Theorem 8 shows that our Libra can reduce the replication factor while keeping good edge-balance.

4. Degree-based Greedy Algorithms

Like the greedy algorithms in PowerGraph (Gonzalez et al., 2012), we can also design greedy algorithms motivated by our degree-based hashing algorithm (Libra).

The greedy graph partitioning (GGP) tries to make effective use of the previous partitioning results to guide the assignment of the current edge, which can be formulated as follows:

$$\operatorname*{argmin}_k \left\{ \sum_{v\in V} |A(v)| \big| M_r, M(e_{r+1}) = k \right\}, \tag{2}$$

$$s.t.\ \max_m |\{e \in E \mid M(e) = m\}| < \lambda\frac{|E|}{p} \quad \text{and} \quad \max_m |\{v \in V \mid Master(v) = m\}| < \rho\frac{|V|}{p},$$

where k is the index number of a machine, $M_r = \{M(e_1), \ldots, M(e_r)\}$ is the assignment of the previous r edges, and e_{r+1} is the current edge to be assigned.

4.1 Algorithms

Now we apply the degree-based strategy to GGP. More specifically, two degree-based greedy algorithms, *Constell* and *Zodiac*, are proposed for GGP.

4.1.1 Constell

Let E_k denote the current set of edges that belong to the kth machine, and let $\mathbf{1}\{condition\}$ denote the indicator function on some condition. Given an edge $e = (v_i, v_j)$, we first define a score for the kth machine as follows:

$$\begin{aligned} Score(k) &= \mathbf{1}\{k \in A(v_i)\} + \mathbf{1}\{k \in A(v_j)\} + \mathbf{1}\{k \in A(v_i), d_i \leq d_j\} \\ &\quad + \mathbf{1}\{k \in A(v_j), d_j \leq d_i\} + balance(k), \\ balance(k) &= \frac{maxedges - |E_k|}{maxedges - minedges + 1}, \end{aligned}$$

where $maxedges = \max_{k \in [p]}\{|E_k|\}$ and $minedges = \min_{k \in [p]}\{|E_k|\}$.

Then, the edge e will be assigned to P_{idx} where idx is decided by:

$$idx = \operatorname*{argmax}_{k \in [p]}\{Score(k)\}.$$

We can find that the above partitioning algorithm adopts the following principles:

- Rule 1: If $A(v_i) \cap A(v_j) \neq \emptyset$, then assign the edge to the least loaded machine in $A(v_i) \cap A(v_j)$.
- Rule 2: If $A(v_i) \cap A(v_j) = \emptyset$ and $A(v_i) \neq \emptyset, A(v_j) \neq \emptyset$, then assign the edge to the least loaded machine in $A(v_L)$, where $L = \operatorname*{argmin}_l \{d_l | l \in \{i, j\}\}$.
- Rule 3: If one of $A(v_i)$ and $A(v_j)$ is not empty, then assign the edge to the least loaded machine in $A(v_L)$, where $A(v_L) = A(v_i) \cup A(v_j)$, which is the non-empty set in $A(v_i)$ and $A(v_j)$.
- Rule 4: If $A(v_i) = \emptyset$ and $A(v_j) = \emptyset$, then assign the edge to the least loaded one of the p machines.

We call this degree-based greedy algorithm *Constell* because it is an enhanced greedy version of *Libra*. The detailed algorithm is presented in Algorithm 2. Note that $balance(k) < 1$, which means the priority of edge-balance is lower than that of minimizing the replication factor in Constell.

4.1.2 Zodiac

The algorithm *Constell* may cause much communication for partitioning because of the synchronization (see Algorithm 2). Inspired by PowerLyra, we introduce a threshold to differentiate vertices with lower degree and higher degree. We separate the whole partitioning algorithm into two steps. In the first step, the edges associated with two low-degree vertices are assigned synchronously, and then the remaining edges are assigned obliviously (i.e., no communication across the machines) in the second step. In particular, for the second step, each machine maintains its own $\{E_j | j \in [p]\}$ and $A(\cdot)$. We denote $\{E_j | j \in [p]\}$ and $A(\cdot)$ of the kth machine as $\{E^k_j | j \in [p]\}$ and $A^k(\cdot)$. Intuitively, the threshold should be at least p (i.e., the number of machines). The larger the threshold is, the slower the partitioning

Algorithm 2 Degree-based Greedy Algorithm I (Constell)

Input: The set of edges E; the set of vertices V; the number of machines p.
Output: The assignment $M(e) \in [p]$ of each edge e.

```
1:  Count the degree d_i of each v_i, ∀i ∈ [n] in parallel        ▷ Initialization
2:  for all k ∈ [p] do
3:      E_k ← ∅
4:  end for
5:  for all e = (v_i, v_j) ∈ E do
6:      Randomly assign each edge to a certain machine
7:  end for
8:  for all k ∈ [p] do                 ▷ Partitioning: synchronously in parallel
9:      for all e = (v_i, v_j) exists now in the kth machine do
10:         M(e) ← argmax_{s∈[p]} {Score(s)}
11:         A(v_i) ← A(v_i) ⋃ {M(e)}
12:         A(v_j) ← A(v_j) ⋃ {M(e)}
13:         E_{M(e)} ← E_{M(e)} ⋃ e
14:     end for
15: end for
```

is. Moreover, the performance is more similar to that of *Constell* as the threshold becomes bigger.

We call this algorithm *Zodiac* because it is a communication-efficient version of *Constell*. The detailed procedure is given in Algorithm 3.

5. Empirical Evaluation

In this section, we conduct empirical evaluation on synthetic and real graphs to verify the effectiveness of our PowerLore framework[1] which includes one degree-based hashing algorithm (Libra) and two degree-based greedy algorithms (Constell and Zodiac). The cluster for evaluation contains 64 nodes (machines). Each node is a 24-core server with 2.2GHz Intel Xeon E5-2430 processor and 96GB of RAM. The machines are connected via 1 GB Ethernet.

5.1 Datasets

The graph datasets used in our experiments include both synthetic and real-world power-law graphs. Each synthetic power-law graph is generated by a combination of two synthetic directed graphs. The in-degree and out-degree of the two directed graphs are sampled from the power-law degree distributions with different power parameters α and β, respectively. The whole collection of the synthetic graphs are separated into two subsets: one subset with parameter $\alpha \geq \beta$ which is shown in Table 1 (a), and the other subset with parameter $\alpha < \beta$ which is shown in Table 1 (b). The real-world graphs are shown in Table 1 (c). Some of the

1. The source code can be downloaded from https://github.com/xcgoner/powerlore.

Algorithm 3 Degree-based Greedy Algorithm II (Zodiac)

Input: The set of edges E; the set of vertices V; the number of machines p; the threshold t.

Output: The assignment $M(e) \in [p]$ of each edge e.

```
1: Count the degree d_i of each v_i, ∀i ∈ [n] in parallel          ▷ Initialization
2: for all k ∈ [p] do
3:     E_k ← ∅
4: end for
5: for all e = (v_i, v_j) ∈ E do
6:     Randomly assign each edge to a certain machine
7: end for
8: for all k ∈ [p] do              ▷ Partitioning: Step 1, synchronously in parallel
9:     for all e = (v_i, v_j) exists now in the kth machine do
10:        if d_i ≤ t and d_j ≤ t then
11:            M(e) ← argmax_{s∈[p]} {Score(s)}
12:            A(v_i) ← A(v_i) ⋃ {M(e)}
13:            A(v_j) ← A(v_j) ⋃ {M(e)}
14:            E_{M(e)} ← E_{M(e)} ⋃ e
15:        end if
16:    end for
17: end for
18: for all k ∈ [p] do                                       ▷ Prepare for Step 2
19:    A^k(·) ← A(·)
20:    for all j ∈ [p] do
21:        E_j^k ← E_j
22:    end for
23: end for
24: for all k ∈ [p] do              ▷ Partitioning: Step 2, obliviously in parallel
25:    Asynchronously in parallel
26:    for all e = (v_i, v_j) exists now in the kth machine do
27:        if d_i > t or d_j > t then
28:            M(e) ← argmax_{s∈[p]} {Score(s)}
29:            A^k(v_i) ← A^k(v_i) ⋃ {M(e)}
30:            A^k(v_j) ← A^k(v_j) ⋃ {M(e)}
31:            E^k_{M(e)} ← E^k_{M(e)} ⋃ e
32:        end if
33:    end for
34: end for
```

real-world graphs are the same as those in the experiment of PowerGraph. Some additional real-world graphs in the UF Sparse Matrices Collection (Davis and Hu, 2011) are also used.

(a) Synthetic graphs: $\alpha \geq \beta$

Alias	α	β	$\lvert E \rvert$
S1	2.2	2.2	71,334,974
S2	2.2	2.1	88,305,754
S3	2.2	2.0	134,881,233
S4	2.2	1.9	273,569,812
S5	2.1	2.1	103,838,645
S6	2.1	2.0	164,602,848
S7	2.1	1.9	280,516,909
S8	2.0	2.0	208,555,632
S9	2.0	1.9	310,763,862

(b) Synthetic graphs: $\alpha < \beta$

Alias	α	β	$\lvert E \rvert$
S10	2.1	2.2	88,617,300
S11	2.0	2.2	135,998,503
S12	2.0	2.1	145,307,486
S13	1.9	2.2	280,090,594
S14	1.9	2.1	289,002,621
S15	1.9	2.0	327,718,498

(c) Real-world graphs

Alias	Graph	$\lvert V \rvert$	$\lvert E \rvert$
Tw	Twitter-2010 (Kwak et al., 2010)	42M	1.47B
Arab	Arabic-2005 (Davis and Hu, 2011)	22M	0.6B
Wiki	Wiki (Boldi and Vigna, 2004)	5.7M	130M
LJ	LiveJournal (Mislove et al., 2007)	5.4M	79M
WG	WebGoogle (Leskovec, 2011)	0.9M	5.1M

Table 1: Datasets

5.2 Experimental Setup

In this subsection, the baselines and evaluation metrics are introduced.

5.2.1 Baselines

We compare our PowerLore framework with some representative distributed graph-computing frameworks, including GraphLab (Low et al., 2010), PowerGraph (GraphLab 2) (Gonzalez et al., 2012), and PowerLyra (Chen et al., 2015). Table 2 lists some properties of these frameworks, where L and H for the access pattern in PowerLyra denote low-degree vertices and high-degree vertices respectively. *Hybrid* and *Hybrid_ginger* are two GP algorithms proposed in PowerLyra, where *Hybrid* is a hashing-based algorithm which is non-greedy and *Hybrid_ginger* is a greedy algorithm.

Framework	GraphLab	PowerGraph	PowerLyra	PowerLore
Graph Placement	edge-cut	vertex-cut	hybrid of edge-cut and vertex-cut	vertex-cut
Access Pattern	bi-direction	bi-direction	L: uni-direction / H: bi-direction	bi-direction
Edge-balance	no	yes	yes	yes
Vertex-balance	yes	no	yes	yes

Table 2: Some representative graph partitioning frameworks.

To evaluate the performance of our degree-based hashing algorithm Libra which is non-greedy, we adopt the following non-greedy baselines for comparison:

- *Random* of PowerGraph.
- *Grid* of GraphBuilder[2].
- *Hybrid* of PowerLyra.

To evaluate the performance of our degree-base greedy algorithms Constell and Zodiac, we adopt the *Hybrid_ginger* of PowerLyra for comparison because *Hybrid_ginger* has been proved to outperform other greedy algorithms (Chen et al., 2015).

5.2.2 Evaluation Metric

Replication factor, edge-imbalance, vertex-imbalance, partitioning time, execution time, execution speedup, and workload-imbalance are used as evaluation metrics. The definitions of replication factor, edge-imbalance, and vertex-imbalance are the same as those of the GP model by vertex-cut in Section 2.2. The partitioning time, execution time, execution speedup and workload-imbalance are defined based on the running of *PageRank* algorithm which is forced to run 100 iterations for all evaluations.

The execution speedup is defined as follows:

$$speedup(Alg, Ours) = \frac{\gamma_{Alg} - \gamma_{Ours}}{\gamma_{Alg}} \times 100\%,$$

where Alg is one of the baseline algorithms, γ_{Alg} is the execution time of PageRank for Alg, $Ours$ is one of our proposed algorithms, and γ_{Ours} is the execution time of PageRank for $Ours$.

The workload-imbalance is defined as follows:

$$workload\text{-}imbalance = \frac{\max\limits_{i \in [p]} \gamma^i_{Alg}}{\frac{1}{p} \sum\limits_{i \in [p]} \gamma^i_{Alg}},$$

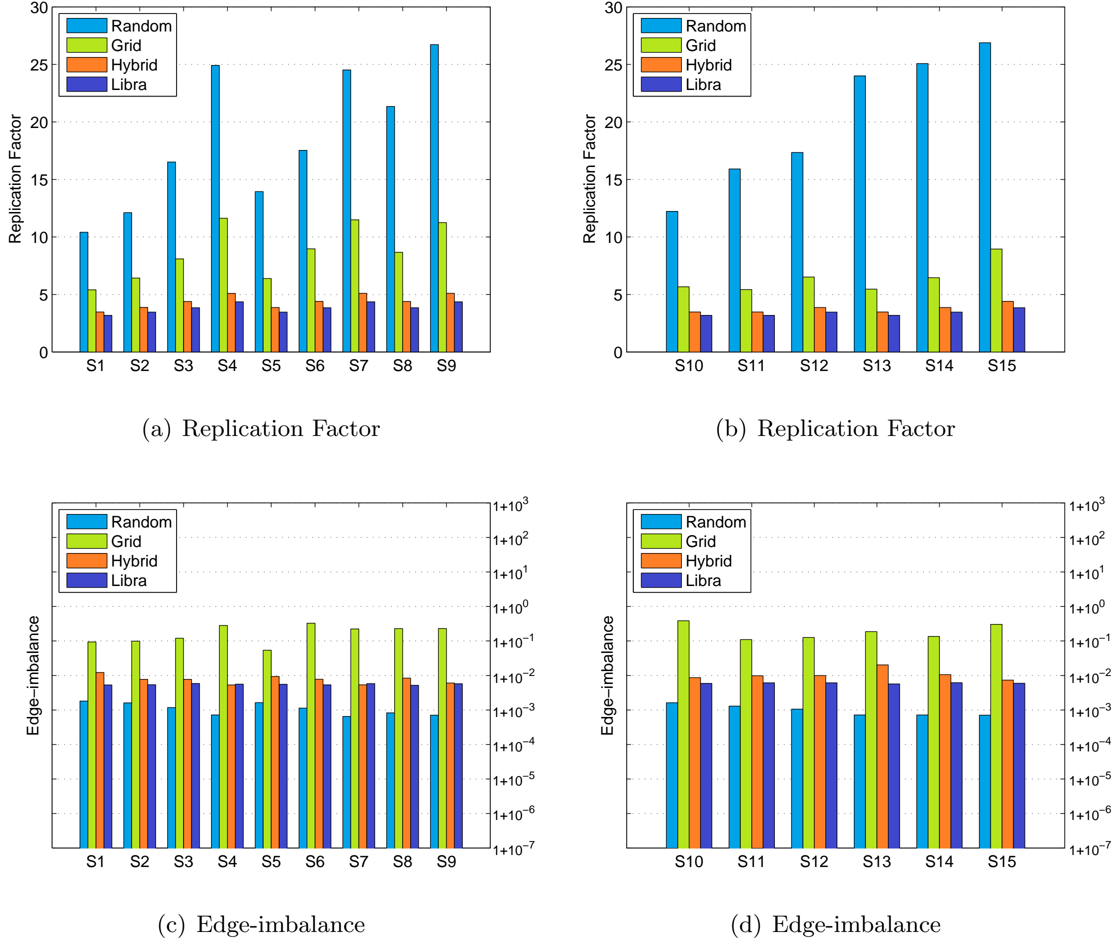
where γ^i_{Alg} is the execution time of some algorithm Alg on the ith machine.

Each experiment is repeated for 20 times, and the average empirical results with standard deviations are reported.

5.3 Experimental Results

We first report the results of non-greedy algorithms, and then report the results of the greedy algorithms.

2. GraphLab 2.2 released in July 2013 has used PowerGraph as its engine, and the *Grid* GP method has been adopted by GraphLab 2.2 to replace the original *Random* GP method. Detailed information can be found at: https://github.com/dato-code/PowerGraph



(a) Replication Factor

(b) Replication Factor

(c) Edge-imbalance

(d) Edge-imbalance

Figure 6: Experimental results on two groups of the synthetic graphs for non-greedy algorithms. The x-axis denotes different datasets in Table 1(a) and Table 1(b). The number of machines is 48.

5.3.1 Non-Greedy Algorithms

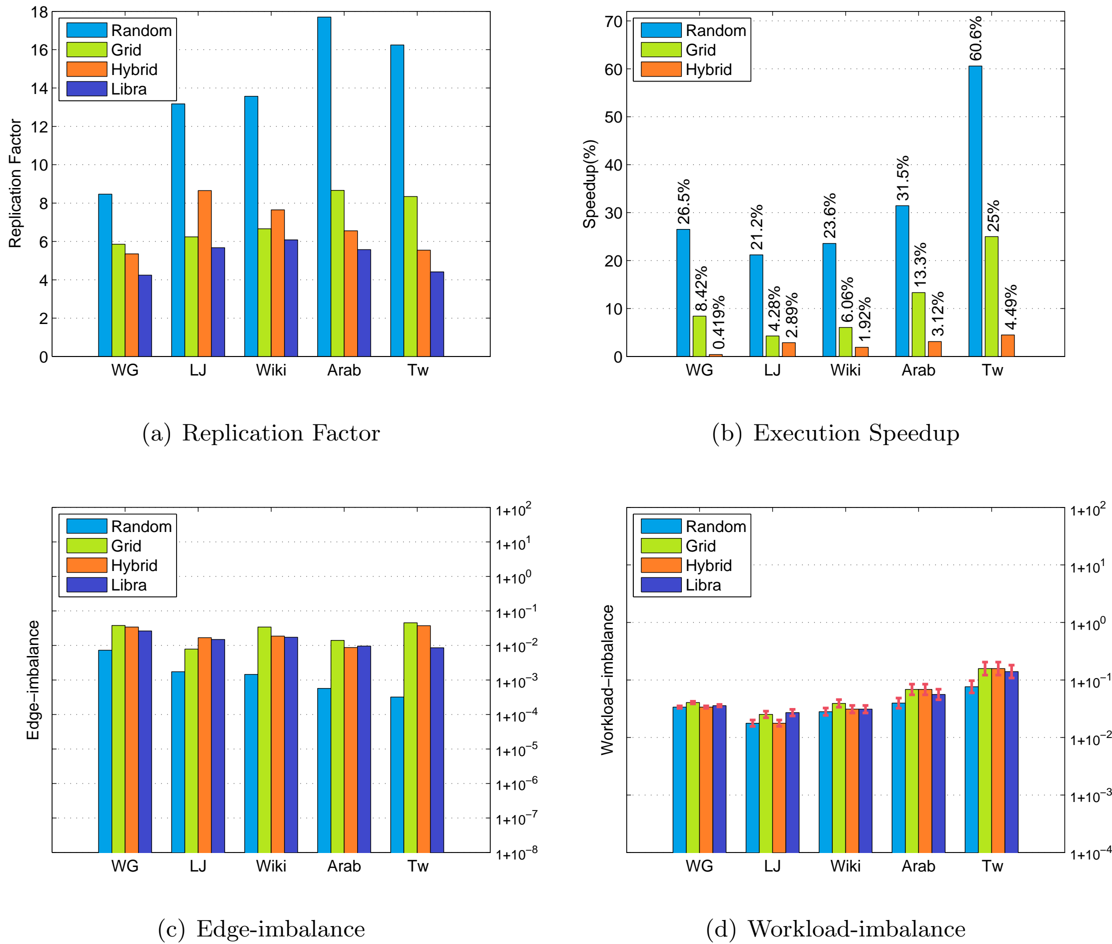
In this section, we compare our degree-based hashing (non-greedy) algorithm Libra with other non-greedy baselines including *Random, Grid* and *Hybrid*. The replication factors on the synthetic datasets are shown in Figure 6(a) and Figure 6(b). We can find that Libra outperforms all the baselines on all graphs. The replication factor is reduced by up to 80% compared to *Random* and 60% compared to *Grid*. Please note that Libra still outperforms *Hybrid* even if *Hybrid* adopts a more complicated strategy to combine both edge-cut and vertex-cut.

The edge-imbalance on the synthetic datasets are shown in Figure 6(c) and Figure 6(d). Libra has better (smaller) edge-imbalance than *Grid* and *Hybrid* in most cases. Although the edge-imbalance of Libra is worse (larger) than that of *Random*, the gap between them

is very small. Please note that *Random* can achieve almost perfectly balanced results when the graph is large enough.

The replication factor for the real-world datasets and speedup of execution time for PageRank are shown in Figure 7(a) and Figure 7(b). In Figure 7(b), *Random, Grid* and *Hybrid* denote $speedup(Random, Libra)$, $speedup(Grid, Libra)$ and $speedup(Hybrid, Libra)$, respectively. We find that Libra is faster than all the other baselines. The speedup of Libra is up to 60% over *Random*, 25% over *Grid* and 4% over *Hybrid*.

The edge-imbalance and workload-imbalance for the real-world datasets are shown in Figure 7(c) and Figure 7(d). We can find that all the methods achieve good (very small) edge-imbalance and workload-imbalance. Libra is slightly better than *Grid* and *Hybrid* in most cases, and is comparable to *Random*.



(a) Replication Factor

(b) Execution Speedup

(c) Edge-imbalance

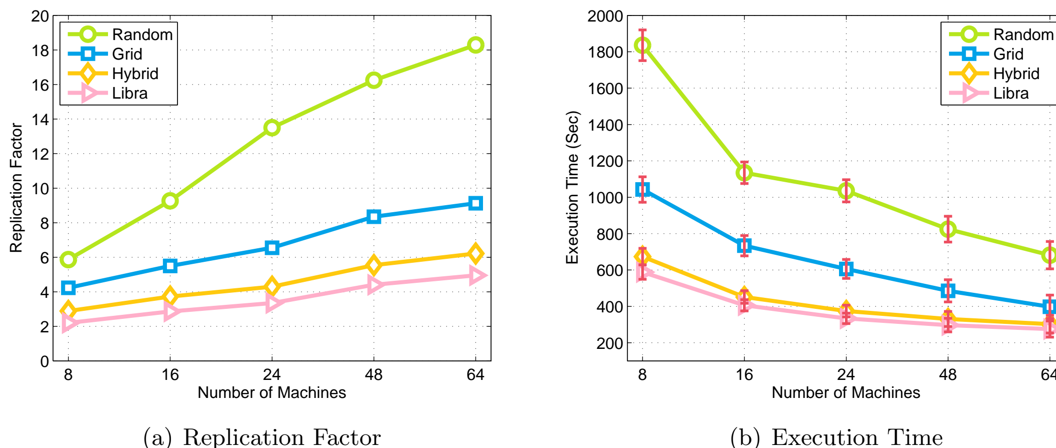
(d) Workload-imbalance

Figure 7: Experimental results on the real-world graphs for non-greedy algorithms. The number of machines is 48.

Sensibility to the number of machines is tested on the Twitter-2010 social network. The replication factor and execution time of PageRank are shown in Figure 8. It can be seen

that our Libra has the smallest replication factors in all cases. Moreover, our Libra has the smallest execution time in all cases. The execution time of Libra is reduced by up to 60% and 20% compared to *Random* and *Grid*, respectively.

Note that for the hashing-based algorithms, the implementation is based on a pseudo-random function. So for the same graph, the partitioning result will never change, which yields no variances for replication factor and edge-imbalance. So we only illustrate the standard deviations for workload-imbalance and execution time.



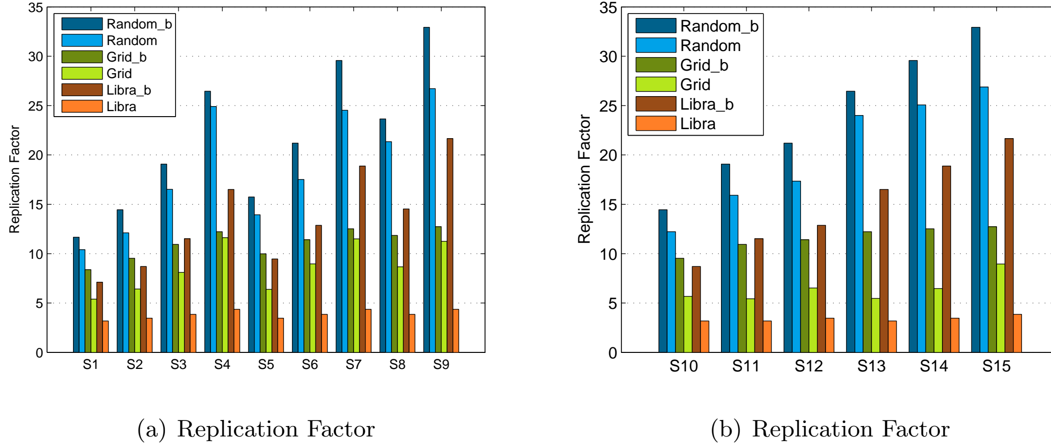
(a) Replication Factor

(b) Execution Time

Figure 8: Experiments on Twitter-2010 for non-greedy algorithms. The number of machines ranges from 8 to 64.

In Figure 9(a) and Figure 9(b), we illustrate the replication factor on the synthetic datasets as well as the upper bounds of the expected replication factor. *Random_b*, *Grid_b* and *Libra_b* denote the upper bounds of the expected replication factors for *Random*, *Grid* and *Libra*, respectively. The power parameter of each synthetic graph is $0.75 \min(\alpha, \beta) + 0.25 \max(\alpha, \beta)$. It seems that the upper bounds are acceptably tight for *Random* and *Grid*, but not tight enough for *Libra*. This is because the upper bound for *Libra* is actually a general upper bound for any edge-hash strategy that is defined via a vertex-hash function. Even if we employ an edge-hash strategy that acts very badly, it shares the same upper bound for the expected replication factor as long as such a strategy is defined via a vertex-hash function.

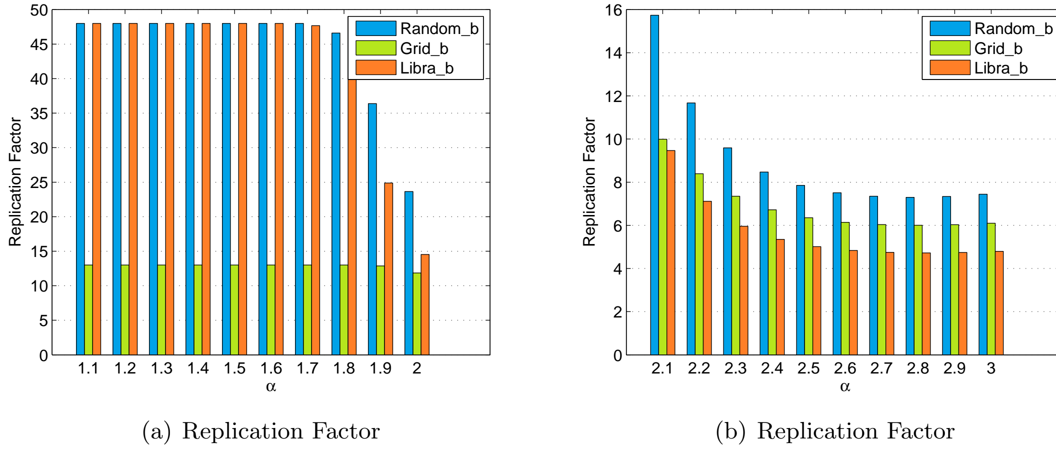
In Figure 10(a) and Figure 10(b), we illustrate the upper bounds of the expected replication factors for degree sequences with power parameter $\alpha \in [1.1, 3]$. There is no guarantee that the upper bound of *Libra* is better than that of *Grid*. Actually, when the expected value of the degrees is very large (e.g., d_{min} is very large or α is very small), the upper bound approaches p for both *Random* and *Libra* while the upper bound of *Grid* approaches $p_1 + p_2 - 1$ (i.e., $6 + 8 - 1 = 13$ when $p = 48$). However, it can be seen that although the upper bound for Libra is very loose, in some cases (e.g., $\alpha > 2$ and d_{min} is small enough) the upper bound is still better than that of *Grid*.



(a) Replication Factor

(b) Replication Factor

Figure 9: Experimental and theoretical results (bounds) on two groups of the synthetic graphs for non-greedy algorithms. The x-axis denotes different datasets in Table 1(a) and Table 1(b). The number of machines is 48.



(a) Replication Factor

(b) Replication Factor

Figure 10: Theoretical results (bounds) for non-greedy algorithms. The x-axis denotes the power parameter. The number of machines is 48.

5.3.2 Greedy Algorithms

In this section, we compare our greedy algorithms Constell and Zodiac to the greedy baseline *Hybrid_ginger*. To show the advantage of greedy algorithms, we also include the results of *Grid*, *Hybrid* and *Libra* which are non-greedy algorithms.

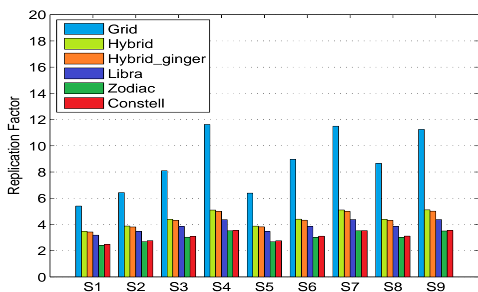
Figures 11 (a)-(h) show the performance on the synthetic graphs with different power-law constants for different algorithms. In all the experiments, the number of machines is 48. Please note that *Libra, Zodiac* and *Constell* in Figure 11 (d) and Figure 11 (h) for

execution speedup denote *speedup*($Hybrid_ginger, Libra$), *speedup*($Hybrid_ginger, Zodiac$) and *speedup*($Hybrid_ginger, Constell$), respectively. We find that the greedy algorithms typically have better performance than the non-greedy counterparts. Furthermore, our greedy algorithms *Zodiac* and *Constell* always have smaller replication factors and better edge-imbalance than all the other algorithms. All algorithms have comparable vertex-imbalance in all cases. In most cases, *Zodiac* has similar performance with *Constell*. An interesting thing is that in some cases, *Zodiac* has comparable or even better performance than *Constell* in both replication factor and speedup. Note that *Zodiac* is a time-efficient version of *Constell* by reducing some synchronization, and the reduction of synchronization is supposed to result in larger replication factors. The reason might be that the greedy algorithm is influenced by the order of the edge sequence. *Zodiac* firstly partitions the vertices with relatively lower degrees, which might result in a better order for the greedy algorithm.

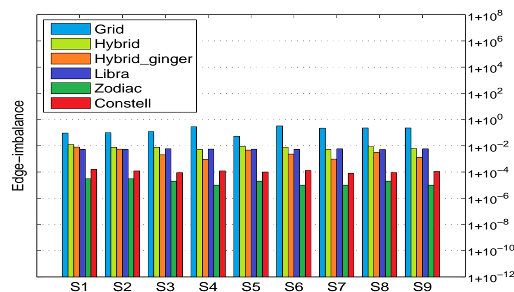
Figures 12 (a)-(f) show the performance on the real-world graphs. *Libra, Zodiac* and *Constell* in Figure 12 (d) for execution speedup denote *speedup*($Hybrid_ginger, Libra$), *speedup*($Hybrid_ginger, Zodiac$) and *speedup*($Hybrid_ginger, Constell$), respectively. In all the experiments, the number of machines is 48. Similar to that on synthetic datasets, the greedy algorithms typically have better performance than the non-greedy counterparts in real-world datasets. Furthermore, our proposed greedy algorithms are also better than the others in terms of both replication factor and edge-imbalance. One interesting thing is that our non-greedy algorithm Libra has better performance than the greedy baseline *Hybrid_ginger* in terms of replication factor (Figure 12 (a)) and execution time (Figure 12 (d)) in most cases. Although the vertex-imbalance of Constell is worse than those of others, the gap is small. The replication factor of Constell is smaller than that of Zodiac. Constell also has better speedup than Zodiac while Zodiac is faster than Constell in terms of partitioning time, as is supposed to be. In terms of partitioning time (Figure 12 (e)), the greedy algorithms are slower than the non-greedy counterparts.

In Figures 13 (a)-(d), we evaluate our algorithms on Twitter-2010 to compare the sensibility and scalability by changing the number of machines. The execution time and partitioning time decrease as the number of machines increases. Although Zodiac has smaller replication factors than Libra, the speedup is small. Constell is the best in both replication factor and execution time, but it is also the slowest in terms of partitioning time. The workload imbalance is good for all methods as the number of machines increases. For both non-greedy and greedy settings, our algorithms achieve smaller replication factor and execution time for the real applications while guaranteeing good workload balance.

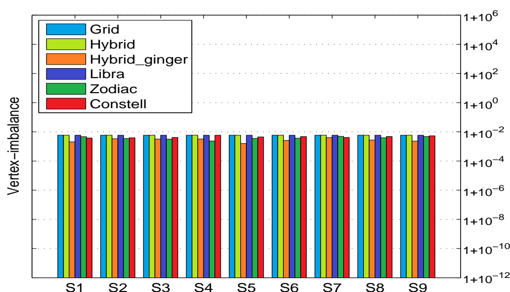
Compared to the baselines, Libra improves the performance while using relatively less time for partitioning. Constell takes the longest time for partitioning while obtaining the shortest execution time. Compared with Constell, Libra costs less time for partitioning and its execution time is only sightly longer. Note that the tradeoff between partitioning time and execution time is inevitable. Therefore, when the learning or mining algorithms based on the DGC frameworks are complicated and time-consuming, Constell should be adopted to speed up the execution time and the partitioning time could be ignored. On the contrary, Libra is good enough for some simple learning or mining algorithms with relatively short execution time. In addition, users can use Zodiac to get a tradeoff between the time cost for executing the application and the time cost for partitioning.



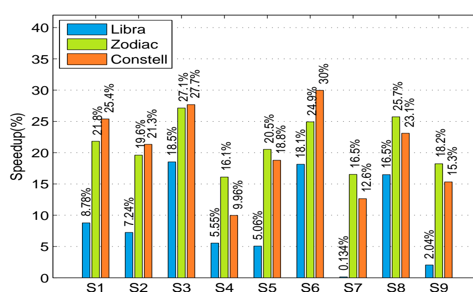
(a) Replication Factor



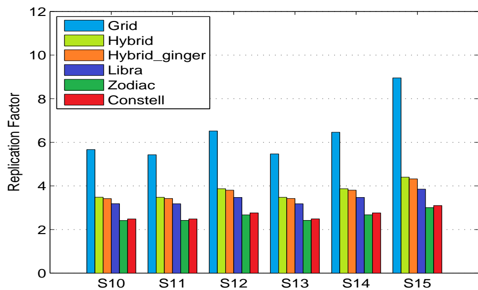
(b) Edge-imbalance



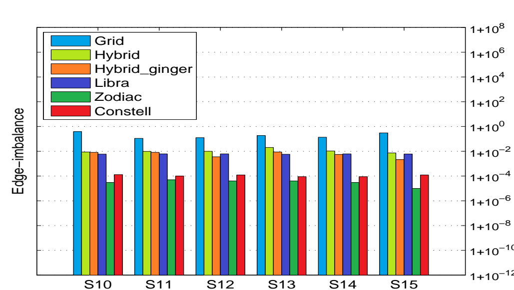
(c) Vertex-imbalance



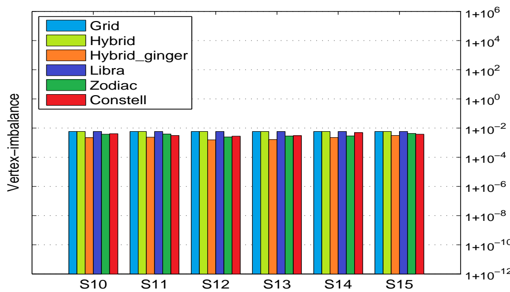
(d) Execution Speedup



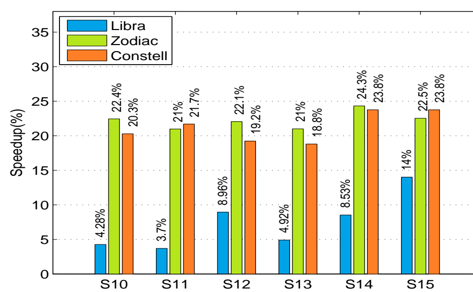
(e) Replication Factor



(f) Edge-imbalance

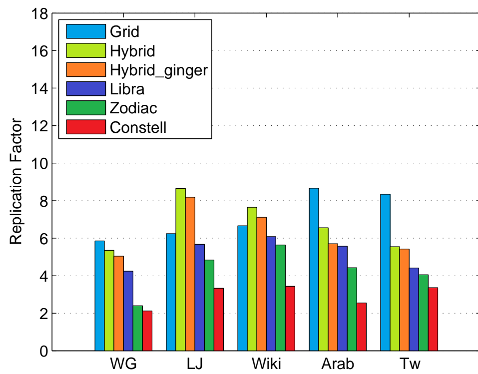


(g) Vertex-imbalance

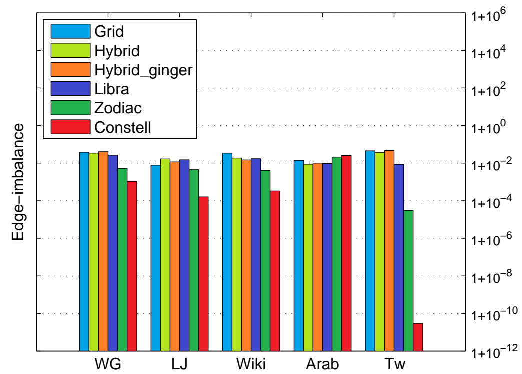


(h) Execution Speedup

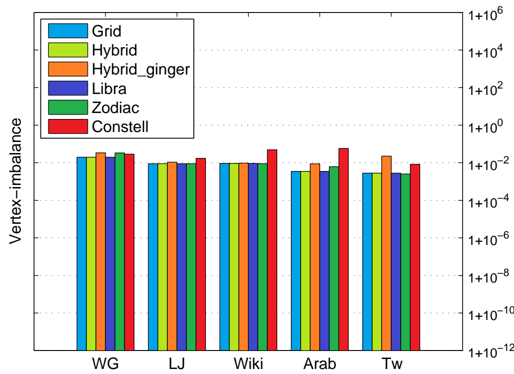
Figure 11: Experiments on two groups of the synthetic graphs. The x-axis denotes different datasets in Table 1(a) and Table 1(b).



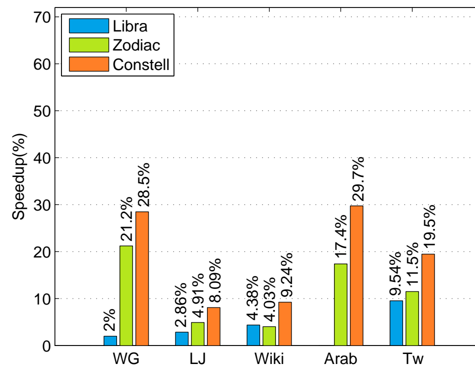
(a) Replication Factor



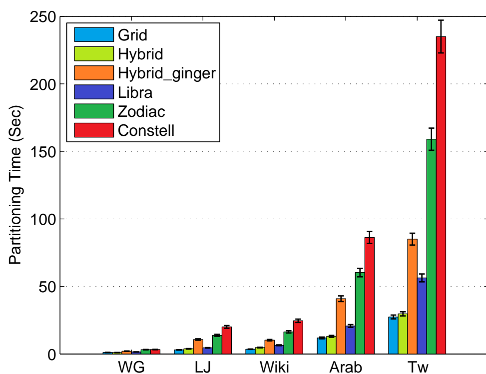
(b) Edge-imbalance



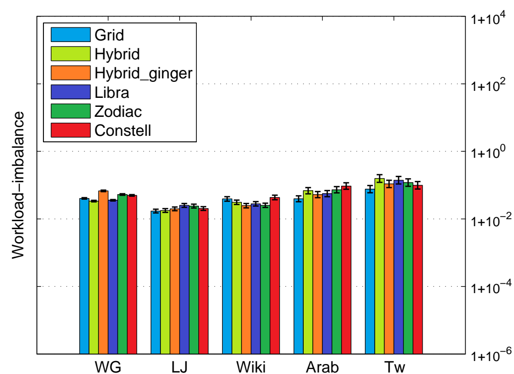
(c) Vertex-imbalance



(d) Execution Speedup



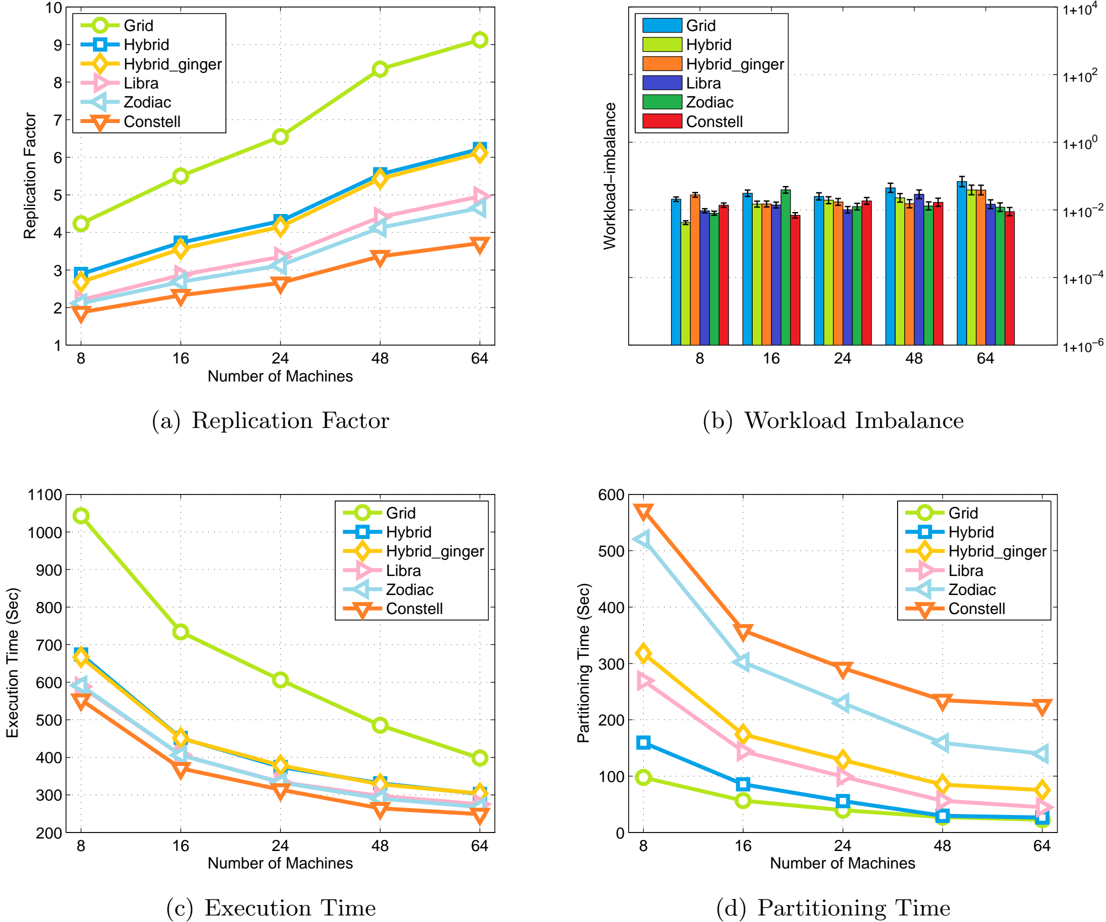
(e) Partitioning Time



(f) Workload-imbalance

Figure 12: Experiments on the real-world graphs.

Please note that for replication factor, edge-imbalance and vertex-imbalance, the standard deviations are very small. So we do not illustrate the standard deviations for them. We only illustrate the standard deviations for partitioning time and workload-imbalance in Figure 12. In Figure 13, the standard deviations for execution time are less than 75 seconds, and the standard deviations for partitioning time are less than 20 seconds. To keep the illustration clean, we do not show the standard deviations for them.



(a) Replication Factor

(b) Workload Imbalance

(c) Execution Time

(d) Partitioning Time

Figure 13: Experiments on Twitter-2010. The number of machines ranges from 8 to 64.

6. Conclusion

In this paper, we have studied vertex-cut graph partitioning in distributed environments. In particular, we have proposed a new graph partitioning framework called *PowerLore* by effectively exploiting the power-law degree distribution in natural graphs. Powerlore includes a degree-base hashing algorithm (Libra) and two degree-based greedy algorithms (Constell and Zodiac). To the best of our knowledge, our work is the first to systematically investigate the degree-based approach for power-law graph partitioning by vertex-cut. Theoretical

analysis shows that Libra can achieve lower communication cost than existing hashing-based methods and can simultaneously guarantee good workload balance. Empirical results on several large power-law graphs show that PowerLore can outperform existing state-of-the-art GP frameworks. In the future work, we will apply our framework to more large-scale machine learning tasks.

Acknowledgments

The authors would like to thank the Action Editor and three anonymous referees for their constructive comments and suggestions on the original version of this paper.

References

Lada A Adamic and Bernardo A Huberman. Zipf's law and the internet. *Glottometrics*, 3 (1):143–150, 2002. 7

Paolo Boldi and Sebastiano Vigna. The webgraph framework I: compression techniques. In *Proceedings of the 13th International Conference on World Wide Web (WWW)*, 2004. 16

Rong Chen, Jiaxin Shi, Yanzhe Chen, and Haibo Chen. PowerLyra: differentiated graph computation and partitioning on skewed graphs. In *Proceedings of the European Conference on Computer Systems*, 2015. 3, 16, 17

Aaron Clauset, Cosma Rohilla Shalizi, and Mark EJ Newman. Power-law distributions in empirical data. *SIAM Review*, 51(4):661–703, 2009. 31

Timothy A Davis and Yifan Hu. The University of Florida sparse matrix collection. *ACM Transactions on Mathematical Software*, 38(1):1, 2011. 15, 16

Joseph E Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2012. 2, 3, 4, 6, 7, 9, 10, 12, 16

Joseph E. Gonzalez, Reynold S. Xin, Ankur Dave, Daniel Crankshaw, Michael J. Franklin, and Ion Stoica. GraphX: Graph processing in a distributed dataflow framework. In *Proceedings of the 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2014. 2

Nilesh Jain, Guangdeng Liao, and Theodore L Willke. Graphbuilder: scalable graph etl framework. In *Proceedings of the First International Workshop on Graph Data Management Experiences and Systems*, 2013. 3, 7, 9, 10, 28

George Karypis and Vipin Kumar. Multilevel graph partitioning schemes. In *Proceedings of the International Conference on Parallel Processing (ICPP)*, 1995. 3

Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is twitter, a social network or a news media. In *Proceedings of the 19th International Conference on World Wide Web (WWW)*, 2010. 7, 16

Aapo Kyrola, Guy E. Blelloch, and Carlos Guestrin. Graphchi: Large-scale graph computation on just a PC. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2012. 2

Jure Leskovec. Stanford large network dataset collection. *URL http://snap. stanford. edu/-data/index. html*, 2011. 16

Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein. GraphLab: A new framework for parallel machine learning. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, 2010. 2, 3, 16

Yucheng Low, Joseph Gonzalez, Aapo Kyrola, Danny Bickson, Carlos Guestrin, and Joseph M. Hellerstein. Distributed graphlab: A framework for machine learning in the cloud. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 2012. 2

Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD)*, 2010. 2, 3, 4

Alan Mislove, Massimiliano Marcon, Krishna P Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and analysis of online social networks. In *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement*, 2007. 16

Isabelle Stanton and Gabriel Kliot. Streaming graph partitioning for large distributed graphs. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, 2012. 3

Charalampos Tsourakakis, Christos Gkantsidis, Bozidar Radunovic, and Milan Vojnovic. Fennel: Streaming graph partitioning for massive scale graphs. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining (WSDM)*, 2014. 3

Lu Wang, Yanghua Xiao, Bin Shao, and Haixun Wang. How to partition a billion-node graph. In *Proceedings of the International Conference on Data Engineering (ICDE)*, 2014. 3

Appendix A. The Proof of Lemma 1

Proof Given a sequence of n vertices $\{v_i\}_{i=1}^n$ and the corresponding degree sequence $D = \{d_i\}_{i=1}^n$, we have:

$$\begin{aligned}
&\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}|A(v_i)|\,\middle|\,D\right] \\
&= \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}\left[|A(v_i)|\,\middle|\,d_i\right] \\
&= \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{p}\Pr(\text{at least one of the adjacent edges of } v_i \text{ is on } P_j) \\
&= \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{p}\big(1-\Pr(\text{none of the adjacent edges of } v_i \text{ is on } P_j)\big) \\
&= \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{p}\big(1-\Pr(\text{one speficic adjacent edge of } v_i \text{ is on } P_j)^{d_i}\big) \\
&= \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{p}\left[1-\left(1-\frac{1}{p}\right)^{d_i}\right] \\
&= \frac{p}{n}\sum_{i=1}^{n}\left[1-\left(1-\frac{1}{p}\right)^{d_i}\right].
\end{aligned}$$

■

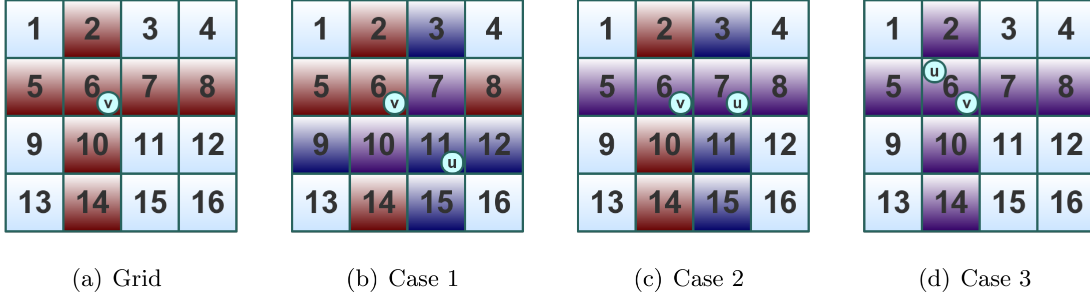
Appendix B. The Proof of Corollary 2

Proof We briefly describe the Grid (Jain et al., 2013) algorithm here to help prove this corollary. Note that we let $(p_1, p_2) = \operatorname{argmin}_{p_1,p_2} |p_1 - p_2|$, where $p_1, p_2 \in \mathbb{N}$ and $p_1 \times p_2 = p$. So, for $\sqrt{p} \in \mathbb{N}$, $p_1 = p_2 = \sqrt{p}$.

The Grid algorithm first constructs a grid with $p_1 \times p_2$ blocks. Each block represents a unique partition. Each vertex is randomly mapped to a certain block via a hash function $grid_hash(v)$. For a certain adjacent edge $e = (v, u)$ of v, the candidate partitions are constrained to the set of blocks which are in the same column or row of the corresponding block $grid_hash(v)$. Note that the edge (v, u) is also an adjacent edge of u, which produces another set of candidate partitions. Then, the edge (v, u) will be randomly assigned to one of the partitions in the intersection of the two sets of candidate partitions.

We use an example in Figure 14 to illustrate how Grid works. In this example, $p = 16$ and we have a grid with 4×4 blocks (i.e., $p_1 = p_2 = 4$). For some vertex v, assume $grid_hash(v) = 6$. So the candidate partitions are the blocks of the column and row in which the 6th block locates, namely, the red shaded blocks $\{2, 5, 6, 7, 8, 10, 14\}$ in Figure 14(a). There are 3 different cases for the location of u:

- $grid_hash(u) \in [16] \setminus \{2,5,6,7,8,10,14\}$. It means that u is hashed to a block which is not in $\{2,5,6,7,8,10,14\}$. For example, in Figure 14(b), $grid_hash(u) = 11$. So the corresponding candidates produced by u are $\{3,7,9,10,11,12,15\}$. Then the intersection of these two candidate sets will be $\{7,10\} = \{2,5,6,7,8,10,14\} \cap \{3,7,9,10,11,12,15\}$. Hence, the edge (v,u) will be randomly assigned to P_7 or P_{10}.
- $grid_hash(u) \in \{2,5,6,7,8,10,14\} \setminus \{6\}$. For example, in Figure 14(c), $grid_hash(u) = 7$. So the corresponding candidates produced by u are are $\{3,5,6,7,8,11,15\}$. Then the intersection of these two candidate sets will be $\{5,6,7,8\} = \{2,5,6,7,8,10,14\} \cap \{3,5,6,7,8,11,15\}$.
- $grid_hash(u) = 6$. For example, in Figure 14(d), $grid_hash(u) = 6$. So the corresponding candidates produced by u are $\{2,5,6,7,8,10,14\}$. Then the intersection of these two candidate sets will be $\{2,5,6,7,8,10,14\}$.



(a) Grid (b) Case 1 (c) Case 2 (d) Case 3

Figure 14: An example for Grid partitioning

For a certain vertex v, there are $2\sqrt{p}-1$ candidate partitions. These candidates can be separated into 3 parts: $p_1 - 1$ candidates in the same column except $grid_hash(v)$, $p_2 - 1$ candidates in the same row except $grid_hash(v)$, and the last one $grid_hash(v)$. For the $p_1 - 1$ candidates in the same column, the probability of an adjacent edge located on one of them is $a_1 = \frac{p_2-1}{p} \times \frac{1}{2} + \frac{p_1-1}{p} \times \frac{1}{p_1} + \frac{1}{p} \times \frac{1}{p_1+p_2-1}$ (check all the 3 cases in Figure 14(b), Figure 14(c), and Figure 14(d)). For the $p_2 - 1$ candidates in the same row, the probability of an adjacent edge located on one of them is $a_2 = \frac{p_1-1}{p} \times \frac{1}{2} + \frac{p_2-1}{p} \times \frac{1}{p_2} + \frac{1}{p} \times \frac{1}{p_1+p_2-1}$. The probability of an adjacent edge located on partition $grid_hash(v)$ is $a_3 = \frac{p_1-1}{p} \times \frac{1}{p_1} + \frac{p_2-1}{p} \times \frac{1}{p_2} + \frac{1}{p} \times \frac{1}{p_1+p_2-1}$ (check the cases in Figure 14(c) and Figure 14(d)). Similar to the proof of Lemma 1, we can get:

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}|A(v_i)|\Big|D\right] = \frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{p}\left(1 - \Pr(\text{none of the adjacent edges of } v_i \text{ is on } P_j)\right)$$

$$= \frac{1}{n}\sum_{i=1}^{n}\left\{(p_1-1)\left[1-\left(1-a_1\right)^{d_i}\right] + (p_2-1)\left[1-\left(1-a_2\right)^{d_i}\right] + \left[1-\left(1-a_3\right)^{d_i}\right]\right\}.$$

■

Appendix C. The Proof of Theorem 3

Proof The probability that vertex v_i has at least one of h_i edges on the jth machine is

$$1 - \Pr(\text{none of the } h_i \text{ edges is on the machine } j) = 1 - \left(1 - \frac{1}{p}\right)^{h_i}.$$

For some vertex v_i, h_i adjacent edges are decentrally hashed and $(d_i - h_i)$ adjacent edges are centrally hashed. Then, the number of replicas of v_i can be separated into two parts:

- For the $(d_i - h_i)$ edges which are centrally hashed, the number of replicas contributed by them is 1 if $h_i < d_i$, and 0 if $h_i = d_i$.
- For the rest h_i adjacent edges which are decentrally hashed, the expected number of replicas on a certain machine is $1 - \left(1 - \frac{1}{p}\right)^{h_i}$, which involves the $p - 1$ machines except for the one that already has a replica.

We define a new sequence of variable $\{g_i\}_{i=1}^n$ such that

$$g_i = \begin{cases} 1 & \text{if } h_i < d_i, \\ 0 & \text{if } h_i = d_i. \end{cases}$$

Putting the two parts together, we have

$$\mathbb{E}\left[|A(v_i)|\right] = g_i + (p-1)\left[1 - \left(1 - \frac{1}{p}\right)^{h_i}\right] = p\left[1 - \left(1 - \frac{1}{p}\right)^{h_i + g_i}\right].$$

Thus, the expected replication factor is:

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}|A(v)|\right] = \frac{1}{n}\sum_{i=1}^{n}\left[p\left(1 - \left(1 - \frac{1}{p}\right)^{h_i+g_i}\right)\right] = \frac{p}{n}\sum_{i=1}^{n}\left[1 - \left(1 - \frac{1}{p}\right)^{h_i+g_i}\right].$$

Note that $h_i \leq d_i$ and $h_i \in \mathbb{N}$, so it is easy to check that $h_i + g_i \leq d_i$ because

$$h_i + g_i = \begin{cases} h_i + 1 \leq d_i & \text{if } h_i < d_i \text{ (or } h_i \leq d_i - 1), \\ h_i + 0 \leq d_i & \text{if } h_i = d_i. \end{cases}$$

By Lemma 1, we obtain

$$\frac{p}{n}\sum_{i=1}^{n}\left[1 - \left(1 - \frac{1}{p}\right)^{h_i+g_i}\right] \leq \frac{p}{n}\sum_{i=1}^{n}\left[1 - \left(1 - \frac{1}{p}\right)^{d_i}\right],$$

which implies that Libra is at least as good as the Random of PowerGraph in terms of the replication factor. ■

Appendix D. The Proof of Theorem 4

Proof Since we assume that the vertices are evenly hashed to all the machines, the h_i adjacent edges of v_i are also evenly assigned to all the machines. Subsequently, each machine has $\frac{h_i}{p}$ edges. Thus, we sum up all the vertices, obtaining $\sum_{i=1}^{n} \frac{h_i}{p}$.

For the rest $d_i - h_i$ adjacent edges of v_i, they are assigned to the same machine. So this part of edges incurs imbalance.

In the above procedure each edge is actually assigned twice. Thus, the final result is

$$\frac{\max_m |\{e \in E \mid M(e) = m\}|}{|E|/p} \approx \frac{\sum_{i=1}^{n} \frac{h_i}{p} + \max_{j \in [p]} \sum_{v_i \in P_j} (d_i - h_i)}{2|E|/p}.$$

■

Appendix E. The Proof of Theorem 5

Proof Note that although the degree distribution of a random graph should take integers, it is common to approximate the power-law degree distribution by continuous real numbers, which is suggested by Clauset et al. (2009).

The discrete power-law distribution is defined as:

$$\Pr(d = x) = \frac{x^{-\alpha}}{B(\alpha, d_{min}, d_{max})}, \qquad \text{for } x \geq x_{min}. \tag{3}$$

$B(\alpha, d_{min}, d_{max})$ is the normalization factor, which is defined as $B(\alpha, d_{min}, d_{max}) = \sum_{x=d_{min}}^{d_{max}} x^{-\alpha}$. For $\alpha > 0$, $d_{max} > d_{min}$, and $d_{min}, d_{max} \in \mathbb{N}$, we have

$$\int_{x=d_{min}}^{d_{max}+1} x^{-\alpha} \mathbf{d}x < B(\alpha, d_{min}, d_{max}) < \int_{x=d_{min}}^{d_{max}+1} (x-1)^{-\alpha} \mathbf{d}x.$$

If $\alpha \in (0, 1) \cup (1, +\infty)$, then

$$\frac{d_{min}^{1-\alpha} - (d_{max}+1)^{1-\alpha}}{\alpha - 1} < B(\alpha, d_{min}, d_{max}) < \frac{d_{min}^{1-\alpha} - d_{max}^{1-\alpha}}{\alpha - 1} + d_{min}^{-\alpha}.$$

If $\alpha = 1$, then

$$\ln(d_{max}+1) - \ln(d_{min}) < B(1, d_{min}, d_{max}) < \ln(d_{max}) - \ln(d_{min}) + d_{min}^{-1}.$$

If $\alpha = 0$, then

$$B(1, d_{min}, d_{max}) = d_{max} - d_{min} + 1.$$

Note that $f(x) = 1 - (1 - \frac{1}{p})^x$ is concave and monotonically increasing w.r.t. x. Taking expectation on the degree sequence $D = \{d_i\}$, we can get

$$\mathbb{E}_D\left[\frac{p}{n}\sum_{i=1}^{n}\left(1-\left(1-\frac{1}{p}\right)^{d_i}\right)\right] = \frac{p}{n}\sum_{i=1}^{n}\mathbb{E}_D\left[1-\left(1-\frac{1}{p}\right)^{d_i}\right]$$
$$= \frac{p}{n}\sum_{i=1}^{n}\mathbb{E}_{d_i}\left[1-\left(1-\frac{1}{p}\right)^{d_i}\right] \leq \frac{p}{n}\sum_{i=1}^{n}\left[1-\left(1-\frac{1}{p}\right)^{\mathbb{E}_{d_i}[d_i]}\right]$$

by Jensen's inequality.

We simply take $d_{max} = n - 1$. Then, when $\alpha \neq 1$ and $\alpha \neq 2$, the expectation is

$$\mathbb{E}_{d_i}[d_i] = \frac{\sum_{x=d_{min}}^{n-1} x^{1-\alpha}}{\sum_{x=d_{min}}^{n-1} x^{-\alpha}} = \frac{B(\alpha-1, d_{min}, n-1)}{B(\alpha, d_{min}, n-1)} < \frac{\alpha-1}{\alpha-2} \times \frac{d_{min}^{2-\alpha} - (n-1)^{2-\alpha} + (\alpha-2)d_{min}^{1-\alpha}}{d_{min}^{1-\alpha} - n^{1-\alpha}}$$

For $\alpha = 1$, it is

$$\mathbb{E}_{d_i}[d_i] = \frac{\sum_{x=d_{min}}^{n-1} x^{0}}{\sum_{x=d_{min}}^{n-1} x^{-1}} = \frac{B(0, d_{min}, n-1)}{B(1, d_{min}, n-1)} < \frac{n - d_{min}}{\ln(n) - \ln(d_{min})}.$$

For $\alpha = 2$, it is

$$\mathbb{E}_{d_i}[d_i] = \frac{\sum_{x=d_{min}}^{n-1} x^{-1}}{\sum_{x=d_{min}}^{n-1} x^{-2}} = \frac{B(1, d_{min}, n-1)}{B(2, d_{min}, n-1)} < \frac{\ln(n-1) - \ln(d_{min}) + d_{min}^{-1}}{d_{min}^{-1} - n^{-1}}.$$

When $n \to +\infty$ and $\alpha > 2$, $\lim_{n\to+\infty} \mathbb{E}_{d_i}[d_i] < \frac{\alpha-1}{\alpha-2} \times d_{min} + \alpha - 1$.

Finally, we have:

$$\mathbb{E}_D\left[\frac{p}{n}\sum_{i=1}^{n}\left(1-\left(1-\frac{1}{p}\right)^{d_i}\right)\right] < p\left[1-\left(1-\frac{1}{p}\right)^{\hat{\Omega}}\right],$$

where $\hat{\Omega} = \begin{cases} \frac{\alpha-1}{\alpha-2} \times \frac{d_{min}^{2-\alpha} - (n-1)^{2-\alpha} + (\alpha-2)d_{min}^{1-\alpha}}{d_{min}^{1-\alpha} - n^{1-\alpha}} & \text{if } \alpha \neq 1 \text{ and } \alpha \neq 2, \\ \frac{n-d_{min}}{\ln(n)-\ln(d_{min})} & \text{if } \alpha = 1, \\ \frac{\ln(n-1)-\ln(d_{min})+d_{min}^{-1}}{d_{min}^{-1}-n^{-1}} & \text{if } \alpha = 2. \end{cases}$ ■

Appendix F. The Proof of Corollary 6

Proof Note that $a_1, a_2, a_3 \in (0, 1)$. So, the following three terms $(p_1 - 1)\left[1 - \left(1 - a_1\right)^x\right]$, $(p_2 - 1)\left[1 - \left(1 - a_2\right)^x\right]$, and $\left[1 - \left(1 - a_3\right)^x\right]$ are also concave and monotonically increasing

w.r.t. x. Similar to the proof of Theorem 5, by using Jensen's inequality we have:

$$\begin{aligned}
&\mathbb{E}_D\left[\frac{1}{n}\sum_{i=1}^{n}\left\{(p_1-1)\left[1-\left(1-a_1\right)^{d_i}\right]+(p_2-1)\left[1-\left(1-a_2\right)^{d_i}\right]+\left[1-\left(1-a_3\right)^{d_i}\right]\right\}\right]\\
&\leq \frac{1}{n}\sum_{i=1}^{n}\left\{(p_1-1)\left[1-\left(1-a_1\right)^{\mathbb{E}_D[d_i]}\right]+(p_2-1)\left[1-\left(1-a_2\right)^{\mathbb{E}_D[d_i]}\right]+\left[1-\left(1-a_3\right)^{\mathbb{E}_D[d_i]}\right]\right\}\\
&\leq (p_1-1)\left[1-\left(1-a_1\right)^{\hat{\Omega}}\right]+(p_2-1)\left[1-\left(1-a_2\right)^{\hat{\Omega}}\right]+\left[1-\left(1-a_3\right)^{\hat{\Omega}}\right],
\end{aligned}$$

where $\hat{\Omega}$ is the same as that in Theorem 5. ■

Appendix G. The Proof of Theorem 7

Proof We take the expectation on $H=\{h_i\}_{i=1}^n$ conditional on $D=\{d_i\}_{i=1}^n$, leading to

$$\mathbb{E}_H\left[1-\left(1-\frac{1}{p}\right)^{h_i+g_i}\Big|D\right]\leq 1-\left(1-\frac{1}{p}\right)^{\mathbb{E}_H\left[h_i+g_i\big|D\right]}.$$

Note that for any specific edge $e=(u,v)$, the edge must be centrally hashed by one of $\{u,v\}$, and be decentrally hased by the another, which implies that the total number of centrally hashed edges must equal the total number of decentrally hashed edges. Furthermore, when $D=\{d_i\}$ are fixed, we have $\sum_{i=1}^n h_i=\sum_{i=1}^n(d_i-h_i)=(\sum_{i=1}^n d_i)-(\sum_{i=1}^n h_i)$, which shows that

$$\sum_{i=1}^{n}h_i=\frac{1}{2}\sum_{i=1}^{n}d_i.$$

Thus, we have

$$\mathbb{E}\left[h_i\right]=\lim_{n\to+\infty}\frac{1}{n}\sum_{i=1}^{n}h_i=\frac{1}{2}\lim_{n\to+\infty}\frac{1}{n}\sum_{i=1}^{n}d_i=\frac{1}{2}\mathbb{E}\left[d_i\right].$$

Because $g_i\leq 1$, we can obtain the following result:

$$\mathbb{E}\left[\frac{p}{n}\sum_{i=1}^{n}\left(1-\left(1-\frac{1}{p}\right)^{h_i+g_i}\right)\right]\leq\frac{p}{n}\sum_{i=1}^{n}\left[1-\left(1-\frac{1}{p}\right)^{\mathbb{E}[h_i+g_i]}\right]\leq\frac{p}{n}\sum_{i=1}^{n}\left[1-\left(1-\frac{1}{p}\right)^{\hat{\Omega}/2+1}\right].$$

Note that the above upper bound is loose. Even if we use an edge-hash function that always chooses to cut a vertex with relatively lower degree (which is exactly contrary to *Libra*), i.e.,

$$edge_hash'(v_i,v_j)=\begin{cases}vertex_hash(v_i) & \text{if } d_i>d_j,\\ vertex_hash(v_j) & \text{otherwise,}\end{cases}$$

the expected replication factor has the same upper bound. So the upper bound is actually the same for a family of edge-hash strategies. Then, why is our choice better than the

others ? To simplify the analysis, we set every $g_i = 1$. Note that $\forall i \in [n]$, $h_i \in [0, d_i]$ and $\sum_{i=1}^{n} h_i = \frac{1}{2}\sum_{i=1}^{n} d_i$. So, when each d_i is fixed and $d_1 \leq d_2 \leq \ldots \leq d_n$, to minimize $\frac{p}{n}\sum_{i=1}^{n}\left[1-\left(1-\frac{1}{p}\right)^{h_i+1}\right]$, the optimal sequence $H = \{h_i\}_{i=1}^{n}$ will be

$$h_n = d_n, h_{n-1} = d_{n-1}, \ldots, h_{j+1} = d_{j+1}, h_j = \frac{1}{2}(\sum_{i=1}^{n} d_i) - (\sum_{i=j+1}^{n} h_i), h_{j-1} = 0, \ldots, h_1 = 0,$$

where $j \in [n]$ is the largest number such that $\frac{1}{2}(\sum_{i=1}^{n} d_i) - (\sum_{i=j+1}^{n} h_i) \leq d_j$ and $\forall i > j, h_i = d_i$. However, such an optimal solution of $\{h_i\}_{i=1}^{n}$ can not be achieved in general. We can only let the sequence close to the optimal one. That is, for vertices with relatively large degrees, let h_i close to d_i, and for vertices with relatively small degrees, let h_i close to 0, which is exactly what *Libra* does.

We define $b_i = \Pr(\text{an adjacent edge of } v_i \text{ is "decentrally hashed"})$. Then, we have

$$b_i = \Pr(d < d_i) + \frac{1}{2} \times \Pr(d = d_i) < \frac{B(\alpha, d_{min}, d_i)}{B(\alpha, d_{min}, n-1)} < \frac{d_{min}^{1-\alpha} - d_i^{1-\alpha} + (\alpha-1)d_{min}^{-\alpha}}{d_{min}^{1-\alpha} - n^{1-\alpha}}.$$

And,

$$\mathbb{E}_H\left[h_i + g_i \middle| D\right] = \mathbb{E}_H\left[h_i \middle| D\right] + \mathbb{E}_H\left[g_i \middle| D\right] = d_i b_i + 1 - b_i^{d_i} \leq d_i.$$

Note that when $\alpha > 1$, for $d_i \ll n$, $\lim_{n\to+\infty} \frac{d_{min}^{1-\alpha} - d_i^{1-\alpha} + (\alpha-1)d_{min}^{-\alpha}}{d_{min}^{1-\alpha} - n^{1-\alpha}} = \frac{d_{min}^{1-\alpha} - d_i^{1-\alpha} + (\alpha-1)d_{min}^{-\alpha}}{d_{min}^{1-\alpha}}$. And $\frac{d_{min}^{1-\alpha} - d_i^{1-\alpha} + (\alpha-1)d_{min}^{-\alpha}}{d_{min}^{1-\alpha}}$ increases as α increases. When $0 < \alpha \leq 1$, for $d_i \ll n$, $\lim_{n\to+\infty} b_i = 0$. Thus, when n is large enough and d_i is relatively small, we can state that b_i is monotonically increasing w.r.t. α. Therefore, as α decreases, h_i will be more likely to close to the optimal sequence, which means more replicas will be expected to be reduced when α decreases.

■

Appendix H. The Proof of Theorem 8

Proof Define $x_i = (d_i - h_i)$. If $\{d_i\}_{i=1}^{n}$ are fixed, then $x_i \in [0, d_i]$. For a specific $j \in [p]$, by Hoeffding's tail inequality, we have the following inequality conditional on the sequence $D = \{d_i\}_{i=1}^{n}$:

$$\Pr_H\Big\{\sum_{v_i \in P_j} x_i \geq \mathbb{E}_H\Big[\sum_{v_i \in P_j} x_i\Big] + t \big| D\Big\} \leq \exp\left(\frac{-2t^2}{\sum_{v_i \in P_j} d_i^2}\right), \quad \text{for any } t > 0.$$

Thus we obtain the inequality of the maximum value of the p individual sums:

$$\Pr_H\Big\{\max_{j\in[p]} \sum_{v_i \in P_j} x_i \leq \frac{n}{p}\mathbb{E}_H[x_i] + t \big| D\Big\} \geq \prod_{j\in[p]}\left[1 - \exp\Big(\frac{-2t^2}{\sum_{v_i \in P_j} d_i^2}\Big)\right].$$

Here we take $t = \epsilon \mathbb{E}_D\Big[\sum_{v_i \in P_j} d_i\Big] = \epsilon \frac{n}{p}\mathbb{E}\big[d\big]$ where ϵ is a small constant value. Then we have

$$\sum_{v_i \in P_j} \frac{d_i^2}{t^2} = \sum_{v_i \in P_j} \frac{d_i^2}{\mathbb{E}^2[d]\epsilon^2 n^2/p^2}.$$

Note that for all $v_i \in P_j$, $\{d_i\}$ are also a sequence of values under power-law distribution when $p \ll n$. When $n \to \infty$, the number of any specific value of degree $\#d_i \backsimeq n \Pr(d_i)$. Thus we have

$$\lim_{n\to\infty} \sum_{v_i \in P_j} \frac{d_i^2}{\mathbb{E}^2[d]\epsilon^2 n^2} \backsimeq \lim_{n\to\infty} \sum_{d=d_{min}}^{n-1} \frac{d^2 n \Pr(d)}{\mathbb{E}^2[d]\epsilon^2 n^2}.$$

Under our assumption $\alpha > 0$, we have:

$$\lim_{n\to\infty} \sum_{d=d_{min}}^{n-1} \frac{d^2 n \Pr(d)}{\mathbb{E}^2[d]\epsilon^2 n^2} = 0.$$

It is easy to check that

$$\sum_{d=d_{min}}^{n-1} \frac{d^2 n \Pr(d)}{\mathbb{E}^2[d]\epsilon^2 n^2} = \frac{\frac{\sum_{x=d_{min}}^{n-1} x^{2-\alpha}}{\sum_{x=d_{min}}^{n-1} x^{-\alpha}}}{\epsilon^2 n \left(\frac{\sum_{x=d_{min}}^{n-1} x^{1-\alpha}}{\sum_{x=d_{min}}^{n-1} x^{-\alpha}}\right)^2} = \frac{\left(\sum_{x=d_{min}}^{n-1} x^{2-\alpha}\right)\left(\sum_{x=d_{min}}^{n-1} x^{-\alpha}\right)}{\epsilon^2 n \left(\sum_{x=d_{min}}^{n-1} x^{1-\alpha}\right)^2}.$$

For $\alpha > 1$ and $\alpha \neq 2$, we have that

$$\begin{aligned}&\sum_{d=d_{min}}^{n-1} \frac{d^2 n \Pr(d)}{\mathbb{E}^2[d]\epsilon^2 n^2} \\ &\leq \frac{(\alpha-2)^2}{(\alpha-3)(\alpha-1)\epsilon^2} \times \frac{\left[d_{min}^{3-\alpha} - (n-1)^{3-\alpha} + (\alpha-3)d_{min}^{2-\alpha}\right]\left[d_{min}^{1-\alpha} - (n-1)^{1-\alpha} + (\alpha-1)d_{min}^{-\alpha}\right]}{n\left[d_{min}^{2-\alpha} - n^{2-\alpha}\right]^2}.\end{aligned}$$

To simplify the expression, we rewrite the above inequality as

$$\sum_{d=d_{min}}^{n-1} \frac{d^2 n \Pr(d)}{\mathbb{E}^2[d]\epsilon^2 n^2} \leq \frac{c_0}{\epsilon^2} \times \frac{c_1(n-1)^{3-\alpha} + c_2(n-1)^{1-\alpha} + c_3(n-1)^{4-2\alpha} + c_4(n-1)^0}{c_5 n^1 + c_6 n^{3-\alpha} + c_7 n^{5-2\alpha}},$$

where $c_0, c_1, c_2, c_3, c_4, c_5, c_6$, and c_7 are some constants that do not depend on n.

For $\alpha > 3$, we have that

$$\lim_{n\to\infty} \sum_{d=d_{min}}^{n-1} \frac{d^2 n \Pr(d)}{\mathbb{E}^2[d]\epsilon^2 n^2} \leq \lim_{n\to\infty} \frac{c_0}{\epsilon^2} \times \frac{c_4(n-1)^0}{c_5 n^1} = 0,$$

where $1/\epsilon^2 = o(n)$.

For $2 < \alpha < 3$, we have that

$$\lim_{n\to\infty} \sum_{d=d_{min}}^{n-1} \frac{d^2 n \Pr(d)}{\mathbb{E}^2[d]\epsilon^2 n^2} \leq \lim_{n\to\infty} \frac{c_0}{\epsilon^2} \times \frac{c_1(n-1)^{3-\alpha}}{c_5 n^1} = 0,$$

where $1/\epsilon^2 = o(n^{\alpha-2})$.

For $1 < \alpha < 2$, we have that

$$\lim_{n\to\infty} \sum_{d=d_{min}}^{n-1} \frac{d^2 n \Pr(d)}{\mathbb{E}^2[d]\epsilon^2 n^2} \leq \lim_{n\to\infty} \frac{c_0}{\epsilon^2} \times \frac{c_1(n-1)^{3-\alpha}}{c_7 n^{5-2\alpha}} = 0,$$

where $1/\epsilon^2 = o(n^{2-\alpha})$.

For $0 < \alpha < 1$, we have that

$$\lim_{n\to\infty} \sum_{d=d_{min}}^{n-1} \frac{d^2 n \Pr(d)}{\mathbb{E}^2[d]\epsilon^2 n^2} \leq \lim_{n\to\infty} \frac{c_0}{\epsilon^2} \times \frac{c_3(n-1)^{4-2\alpha}}{c_7 n^{5-2\alpha}} = 0,$$

where $1/\epsilon^2 = o(n)$.

For $\alpha = 2$, we have that

$$\lim_{n\to\infty} \sum_{d=d_{min}}^{n-1} \frac{d^2 n \Pr(d)}{\mathbb{E}^2[d]\epsilon^2 n^2} \leq \lim_{n\to\infty} \frac{(n-d_{min})\left(d_{min}^{-1} - (n-1)^{-1} + d_{min}^{-2}\right)}{\epsilon^2 n \left(\ln(n) - \ln(d_{min})\right)^2} = 0,$$

where $1/\epsilon^2 = o(\ln^2(n))$.

For $\alpha = 1$, we have that

$$\lim_{n\to\infty} \sum_{d=d_{min}}^{n-1} \frac{d^2 n \Pr(d)}{\mathbb{E}^2[d]\epsilon^2 n^2} \leq \lim_{n\to\infty} \frac{\left(\ln(n-1) - \ln(d_{min}) + d_{min}^{-1}\right)\left((d_{min}-1)^{-2} - (n-1)^{-2}\right)}{-2\epsilon^2 n \left(n - d_{min}\right)^2} = 0,$$

where $1/\epsilon^2 = o(n/\ln(n))$.

Thus if n is large enough, under our choice of t, $\left(1 - \exp\left(-2t^2/\sum_{v_i\in P_j} d_i^2\right)\right)$ is nearly 1. That is, $\max_{j\in[p]} \sum_{v_i\in P_j} x_i \leq \frac{n}{p}\mathbb{E}[x_i] + t$ w.h.p. (with high probability) when n is very large. Namely,

$$\max_{j\in[p]} \sum_{v_i\in P_j} x_i \leq \frac{n}{p}\mathbb{E}_H[x_i] + t = \frac{n}{p}\mathbb{E}_H[x_i] + \epsilon\frac{n}{p}\mathbb{E}\left[d\right].$$

By using the result above, the following inequality is satisfied w.h.p:

$$\sum_{i=1}^{n} \frac{h_i}{p} + \max_{j\in[p]} \sum_{v_i\in P_j} (d_i - h_i) \leq \sum_{i=1}^{n} \frac{h_i}{p} + \frac{n}{p}\mathbb{E}_H[x_i] + \epsilon\frac{n}{p}\mathbb{E}\left[d_i\right].$$

Taking the expectation w.r.t. $\{h_i\}$ and $\{d_i\}$, we have:

$$\begin{aligned}
&\mathbb{E}_{H,D}\left[\sum_{i=1}^{n}\frac{h_i}{p}+\max_{j\in[p]}\sum_{v_i\in P_j}(d_i-h_i)\right]\\
&\leq \mathbb{E}_{H,D}\left[\sum_{i=1}^{n}\frac{h_i}{p}\right]+\frac{n}{p}\mathbb{E}_{H,D}[d_i-h_i]+\epsilon\frac{n}{p}\mathbb{E}_{H,D}\left[d\right]\\
&=(1+\epsilon)n\mathbb{E}_{H,D}\left[d\right]/p\\
&=(1+\epsilon)n\mathbb{E}_{D}\left[d\right]/p,
\end{aligned}$$

which completes the proof. ■