Graph Edge Partitioning via Neighborhood Heuristic

Chenzi Zhang*†
University of Hong Kong
czzhang@cs.hku.hk

Fan Wei†
Stanford University
fanwei@stanford.edu

Qin Liu‡
Huawei Noah's Ark Lab
karlqliu@gmail.com

Zhihao Gavin Tang
University of Hong Kong
zhtang@cs.hku.hk

Zhenguo Li§
Huawei Noah's Ark Lab
li.zhenguo@huawei.com

ABSTRACT

We consider the edge partitioning problem that partitions the edges of an input graph into multiple balanced components, while minimizing the total number of vertices replicated (one vertex might appear in more than one partition). This problem is critical in minimizing communication costs and running time for several large-scale distributed graph computation platforms (e.g., PowerGraph, Spark GraphX). We first prove that this problem is NP-hard, and then present a new partitioning heuristic with polynomial running time. We provide a worst-case upper bound of replication factor for our heuristic on general graphs. To our knowledge, we are the first to provide such bound for edge partitioning algorithms on general graphs. Applying this bound to random power-law graphs greatly improves the previous bounds of expected replication factor. Extensive experiments demonstrated that our partitioning algorithm consistently produces much smaller replication factors on various benchmark data sets than the state-of-the-art. When deployed in the production graph engine, PowerGraph, in average it reduces replication factor, communication, and running time by 54%, 66%, and 21%, respectively.

CCS CONCEPTS

• **Information systems** → **Computing platforms**; • **Mathematics of computing** → **Graph algorithms**;

KEYWORDS

Graph edge partitioning; distributed graph mining

*Work performed during the author's internship at Noah's Ark Lab, Huawei.
†Equal contribution.
‡Work performed during the author's graduate study at the Chinese University of Hong Kong.
§Corresponding author.

KDD'17, August 13-17, 2017, Halifax, NS, Canada

ACM ISBN 978-1-4503-4887-4/17/08…$15.00
https://doi.org/10.1145/3097983.3098033

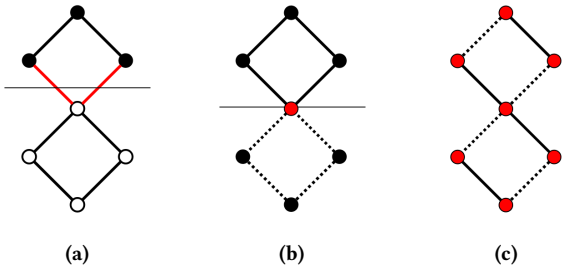

Figure 1: Vertex vs edge partitioning. (a) Ideal vertex partitioning cuts two edges; (b) Ideal edge partitioning cuts only one vertex, Replication Factor=1.14 **(solid edges for one partition, dashed edges for the other); (c) Worst edge partitioning, Replication Factor**=2**.**

1 INTRODUCTION

To handle large-scale graphs, distributed graph engines [11, 12, 16] partition the input graph and parallelize the computation on a cluster of machines. A traditional approach to graph partitioning is *vertex partitioning*: a distributed graph engine partitions the vertex set of a graph into balanced partitions such that the number of edges across different partitions is minimized. However, for most of the large real-world graphs, *edge partitioning* (Fig. 1) is found to be more effective in advanced distributed graph engines [6, 7, 16], which evenly assigns edges to machines in a way that minimizes the number of times each vertex is cut (i.e., replicated). Because most real-world graphs (e.g., social networks, web graphs) have *skewed power-law degree distribution*, which means that most vertices have relatively few neighbors while a few have many neighbors (e.g., celebrities on Facebook's social network). The power-law degree distribution can lead to substantial work imbalance in distributed graph systems that adopt vertex partitioning. Since the storage, communication, and computation complexity of a partition is linear in the number of edges in the partition, the running time of each vertex partition can vary widely. Not surprisingly, researchers demonstrated that edge partitioning performs better on many large real-world graphs [6, 7]. This important finding attracts great interests in edge partitioning recently [3, 6, 13, 15, 17].

Edge partitioning has been widely adopted in recent graph systems including PowerGraph [6], Spark GraphX [7], and Chaos [16], to divide the graph across machines. It turns out that the overall

system performance depends greatly on the quality of graph partitioning. To understand what makes a good edge partitioning, two issues should be addressed. Firstly, the jobs should be distributed evenly across machines, in order to minimize delay caused by stragglers during each round or superstep of computations. More critically, the communication between machines should be minimized, which is often a bottleneck in graph computing due to the intrinsic dependency and expensive random access in the graph [6]. While it is relatively easy to solve the first issue by partitioning the edges evenly, the second issue can be highly challenging, which asks to minimize the so-called *vertex replication* (Section 2.2) that is caused when two adjacent edges (join at the same vertex) are allocated to different machines.

Quite a few methods have been proposed for edge partitioning. Chaos [16] distributes the edges randomly (RAND) under high network bandwidth. RAND ignores all graph structure and incurs high vertex replication (Section 5). The density-based hashing method (DBH) [17] first partitions the vertices randomly, and then assigns each edge (x, y) by following one of its end vertices (i.e., x and y) of smaller degree. DBH can exploit the skewed degree distribution of power-law graphs, but it leverages little graph structure, like RAND. Oblivious [6] is a streaming algorithm that considers the distribution of previously assigned edges when assigning an incoming edge – an edge is more likely to be assigned to the partition with more adjacent edges. HDRF [15] is another streaming algorithm that extends Oblivious by further exploiting the power-law degree distribution, like DBH. Both HDRF and Oblivious rely only on historical data, and thus use the graph structure only partially. Sheep [13] partitions the graph in a divide and conquer manner, which uses more graph structure than Oblivious and HDRF, but it works well only for tree-like graphs. Those methods mentioned above, due to their not attempting to exploit the graph structure, typically yield high replication factor. The notable multi-level vertex partitioning algorithm METIS [8] is extended for edge partitioning [3], which makes full access to the graph structure by partitioning the graph entirely in memory. This does lead to the state-of-the-art replication factors on a great number of graphs [13], but it is not applicable to large graphs.

In this paper, we present a heuristic called *NE (Neighbor Expansion)* that is developed based on a new, theoretically sound partitioning model that greedily maximizes edge locality. Our main contributions are summarized as follows:

- We establish theoretical understanding of the edge partitioning problem, including proving its NP-hardness and showing that every graph can be p-edge partitioned with replication factor $O(\sqrt{p})$.
- We propose a new edge partitioning heuristic called NE, and provide a worst-case upper bound of replication factor for our heuristic on general graphs. To the best of our knowledge, we are the first to provide such bound for edge partitioning algorithms on general graphs. We also show that it improves over existing bounds on random power-law graphs. We conduct extensive experiments which show that it significantly reduces the replication factor.
- We apply our partitioner to a distributed graph engine, PowerGraph. In average it reduces replication factor, communication, and running time by 54%, 66%, and 21%, respectively.

2 GRAPH EDGE PARTITIONING

In this section, we first formalize the graph edge partitioning problem. We then show it is NP-hard and prove a sharp bound of replication factors (Section 2.2) for general graphs.

2.1 Notation

Let $G = (V, E)$ be an undirected, unweighted graph with $n = |V|$ vertices and $m = |E|$ edges. An edge connecting vertices x and y is denoted as (x, y) or $e_{x,y}$. For a vertex x, denote by $N(x) = \{y|(x, y) \in E\}$ the set of vertices adjacent to x. For a vertex set S, denote by $N(S) = \cup_{x\in S} N(x)$ and $E_S = \{(x, y) \in E : x, y \in S\}$. For an integer $p \geq 2$, denote by $[p] = \{1, 2, \ldots, p\}$. For an edge set E_i, denote $V(E_i) = \{x|\{x, y\} \in E_i\}$ as the set of vertices covered by E_i.

2.2 Problem Statement

A p-edge partitioning of G refers to a disjoint partitioning of its edge set E into p subsets E_i, such that $E_i \subseteq E, \cup_{i\in[p]} E_i = E$, and $E_i \cap E_j = \emptyset$ for $i \neq j$. A p-edge partitioning is α-balanced if

$$\max_{i\in[p]} \{|E_i|\} \leq \lceil \alpha |E|/p \rceil. \tag{1}$$

The *replication factor* [3] of a partitioning is defined as

$$\mathrm{RF}(E_1, \ldots, E_p) := \frac{1}{|V|} \sum_{i\in[p]} |V(E_i)|. \tag{2}$$

Definition 1 (MIN-RF(p, α)). *The MIN-RF(p, α) problem is to find an α-balanced p-edge partitioning $\{E_i\}_{i\in[p]}$ to minimize the replication factor RF$(E_1, \ldots, E_p)$.*

2.3 NP-Hardness

The p-edge partitioning problem has been proved to be NP-hard in [3] when p grows with $n = |V|$. To our best knowledge, it has not been proved elsewhere that this problem is NP-hard when p is constant[1]. We fill this gap and give the following NP-hardness result (proof in Appendix A).

Theorem 2.1. *For any $\alpha \geq 1$, $p \geq 2$, the MIN-RF(p, α) problem is NP-hard with respect to the number of vertices. In particular, the MIN-RF$(2, \alpha)$ problem is NP-hard.*

2.4 Bound for General Graphs

In this section, we show that it is always possible to achieve $O(\sqrt{p})$ replication factor for any MIN-RF(p, α) problem (though it may take exponentially long time to find such a partitioning). We also show that this bound is tight up to constant factors. These results mean that $O(\sqrt{p})$ is the best replication factor we can hope for if we are dealing with general graphs. However, for certain graphs like power-law graphs, it is possible to obtain a tighter bound (Section 4.2).

[1] In [3], the NP-hardness is proved by a reduction from 3-partition problem. When p is constant, the 3-partition problem, hence the derived edge partitioning problem, is polynomial time solvable.

Lemma 2.2. *For any graph G and any positive integer p, there always exists a 1-balanced p-edge partitioning such that the replication factor is at most $2\sqrt{p} + \frac{p}{n}$, which is $2\sqrt{p} + o(1)$ if $p = o(n)$.*

Proof. We first prove the following claim.

Claim 1. *Given any graph H with t vertices and m edges, and let k be a positive integer, there must exist a subset of vertices U such that $|U| \le \frac{t-1}{\sqrt{k}} + 1$, and the number of edges in the induced subgraph by U is at least m/k.*

Proof. We use an averaging argument. Consider all subsets of size $l = \frac{t-1}{\sqrt{k}} + 1$. If none of them is such that the induced graph on the vertex set has more than m/k edges, we would have $\sum_{U \subset V(H):|U|=l} |E(U)| \le m/k \cdot \binom{t}{l}$. On the other hand, each edge in H is counted in $\binom{t-2}{l-2}$ vertex sets of cardinality l. Therefore we have $\sum_{U \subset V(H):|U|=l} |E(U)| = m \cdot \binom{t-2}{l-2}$. Combining the two formulas above, we have

$$m/k \cdot \binom{t}{l} \ge m \cdot \binom{t-2}{l-2},$$

which is equivalent to $t(t-1)/l(l-1) \ge k$. However, $l > t/\sqrt{k}$, which means

$$\frac{t(t-1)}{l(l-1)} < \left(\frac{t-1}{l-1}\right)^2 = k. \qquad \square$$

Given the claim, in $G_0 = G$, we can find a vertex set U_1 such that the number of edges induced by U_1, i.e., the number of edges in $G_0[U_1]$ is at least $|E(G_0)|/p$. By the claim, we know $|U_1| \le \frac{n-1}{\sqrt{p}} + 1$.

Then we remove $|E(G_0)|/p$ edges in $G_0[U_1]$ from G_0, obtaining G_1. By a similar argument, we can find a vertex set U_2 such that the number of edges induced by U_2, i.e., the number of edges in $G_1[U_2]$ is at least $|E(G_1)|/(p-1) = |E(G)|/p$. By the claim, we know $|U_2| \le \frac{n-1}{\sqrt{p-1}} + 1$. Removing $|E(G_1)|/(p-1)$ edges in $G_1[U_2]$ from G_1, obtaining G_2. Repeating this process, eventually we have obtained $U_1, U_2, \ldots, U_p$ that exhaust all the edges in G, while

$$|U_i| \le \frac{n-1}{\sqrt{p-(i-1)}} + 1.$$

Therefore the replication factor is

$$\sum_{1 \le i \le p} |U_i|/n \le \frac{1}{n} \sum_{1 \le i \le p} \left(\frac{n-1}{\sqrt{p-(i-1)}} + 1 \right) \le 2\sqrt{p} + \frac{p}{n}. \qquad \square$$

Remark 1. *For a graph, the lemma gives a 1-balanced p-partitioning, which is also a valid partitioning of MIN-RF(p, α) problem for $\alpha \ge 1$.*

Lemma 2.3. *There exists a graph such that the replication factor is at least $\sqrt{p/2}\sqrt{(n-1)/n} = \sqrt{p/2}(1 - o_n(1))$ for any 2-balanced partitioning.*

The proof also generalizes to α-partitioning for any $\alpha \ge 1$.

Proof. Consider a complete graph on n vertices. After partitioning into p parts, we have the vertex sets being $V_1, \ldots, V_p$. Suppose $|V_i| = n_i$ and the edges in the i-th machine is e_i. Therefore $\binom{n_i}{2} \ge e_i$, which means $n_i \ge \sqrt{2e_i}$. We know $\sum_{i=1}^{p} e_i = \binom{n}{2}$ and $e_i \le 2\binom{n}{2}/p$ by the definition of 2-balancedness. Therefore

$$\sum_{i=1}^{p} n_i \ge \sum_{i=1}^{p} \sqrt{2e_i} \ge p/2 \cdot \sqrt{2 \cdot 2\binom{n}{2}/p} = \sqrt{p/2}\sqrt{n(n-1)},$$

where the last inequality is by Jensen inequality. Thus the replication factor is at least

$$\frac{1}{n}\sum_{i=1}^{p} n_i \ge \frac{1}{n}\sqrt{p/2}\sqrt{n(n-1)} = \sqrt{p/2}\sqrt{(n-1)/n}. \qquad \square$$

3 ALGORITHM

In this section, we propose a new edge partitioning algorithm, which partitions the graph iteratively in p rounds. In each round, one edge set is generated (for one machine). Specifically, in round i, edge set E_i is selected from the working graph G_i containing all un-assigned edges so far, i.e., $G_i = (V, E \setminus \cup_{j<i} E_j)$. Empty initially, E_i is expanded in steps until $|E_i| > \alpha m/p$. In each step, one vertex x is selected according to a neighbor expansion heuristic discussed below. Then, adjacent edges of x are added to E_i, and x is added to *core set* C. *Boundary set* $S = V(E_i)$ is the vertex set covered by E_i. Since edges covered by S are not necessarily in E_i, we also add those edges to E_i without increasing vertex replication. The above step is repeated until $|E_i| > \alpha m/p$. We now elaborate the heuristic to select the core vertex x.

Neighbor Expansion. By construction the core set C is always contained in the boundary set S. In case $S \setminus C = \emptyset$, x is picked randomly from $V \setminus C$. Otherwise, it is selected as follows,

$$x := \arg\min_{v \in S \setminus C} |N(v) \setminus S|.$$

The objective $|N(v) \setminus S|$ is the number of vertices that will be assigned to machine i, if v is selected as core vertex and its adjacent edges are added to E_i. Our heuristic is to select x to minimize the number of vertices to be added to the boundary set S, thereby to minimize the increment of replication factor due to adjacent edges of x.

Example. We illustrate one step of our algorithm in Fig. 2. The left and right figures stand for the graph before and after this step. Allocated edges are denoted in solid line and edges in dashed line have not been allocated yet. At the beginning, there are 2 vertices in the core set C, 4 vertices in the boundary set S, and there are 4 allocated edges. Now we need to choose the next core vertex to expand the core set C. Among the candidates $S \setminus C = \{x, z\}$, vertex x is selected because $|N(x) \setminus S| = 1 < |N(z) \setminus S| = 3$. Then, vertex x is added to C and S, and its neighbor y is added to S. Two edges $e_{x,y}$ and $e_{y,z}$ are allocated in this step.

We summarize our heuristic in Algorithm 1, where for ease of presentation the sub-routine of core set update and boundary set update is described in Algorithm 2.

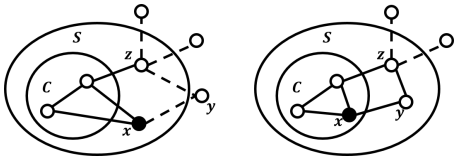

Figure 2: Illustrating a step in our algorithm. Left: a core vertex x is selected. Right: the core vertex set C, the boundary vertex set S, and the edge set E_i are updated.

Algorithm 1 Generate one edge partition E_k.

```
1: procedure EXPAND(E, δ)                          ▷ δ = αm/p
2:     C, S, E_k ← ∅
3:     while |E_k| ≤ δ do
4:         if S \ C = ∅ then
5:             x is selected randomly in V \ C
6:         else
7:             x ← arg min_{v∈S\C} |N(v) \ S|
8:         AllocEdges(C, S, E_k, x)
```

Algorithm 2 Allocate edges for core vertex x.

```
1: procedure AllocEdges(C, S, E_k, x)
2:     C ← C ∪ {x}, S ← S ∪ {x}
3:     for y ∈ N(x) \ S do
4:         S ← S ∪ {y}
5:         for z ∈ N(y) ∩ S do
6:             E_k ← E_k ∪ {e_{y,z}}
7:             E ← E \ {e_{y,z}}
8:             if |E_k| > δ then return
```

4 ANALYSIS

4.1 Upper bound of replication factor

We analyze the replication factor for Algorithm 1. Let $S_k = V(E_k)$ be the covered vertex set when the algorithm terminates for partition k. To the best of our knowledge, we are the first to provide a worst-case upper bound of replication factor on general graphs.

THEOREM 4.1. *Suppose G has no isolated vertex, and let E_i's be the edge sets produced by Algorithm 1 solving MIN-RF(p,α) and $S_i = V(E_i)$. Then, we have*

$$RF = \frac{1}{n}\sum_{i\in[p]}|S_i| \le \frac{1}{n}\left(\frac{3}{2}|E| + \frac{1}{2}\sigma + p - 1\right), \tag{3}$$

where σ is the number of connected components in the original graph.

REMARK 2. *A trivial bound for the replication factor is $\frac{1}{n}\sum_{i\in[p]}|S_i| \le \frac{2|E|}{n}$ as each edge corresponds to 2 vertices. To our best knowledge, our bound is the first improvement for general graphs that brings down the constant from 2 to 3/2.*

PROOF. Observe that Alg. 1 contains p rounds of iterations. In the i-th iteration, a new vertex in $S_i \setminus C_i$ is added to the core C_i and some edges are added to E_i until $|E_i| > \frac{\alpha m}{k}$. Recall $S_i = V(E_i)$.

We analyze how these variables change as the number of steps t increase. We rename the variables and sets above in terms of t for convenience. Let $s_0 = 0$. At each step $t \ge 1$, a new vertex is added to the core C_{s_t}. Thus $1 \le s_t \le p$. Let $C_{i,t}, S_{i,t}$, and $E_{i,t}$ be the sets C_i, S_i, E_i at the beginning of step t respectively. Let σ_t be the number of connected components of $G_t = G \setminus \bigcup_{j=1}^{p} E_{j,t}$ removing the isolated vertices. Let $|G_t|$ be the number of edges in G_t.

We define a potential function $\Phi_t = \frac{3}{2}|G_t| + \frac{1}{2}\sigma_t + p - s_t + \sum_{i\in[p]}|S_{i,t}|$. Initially, all $S_{i,0} = \emptyset$. We will show that when the algorithm terminates at $t = T$, $\Phi_T \le \Phi_0$. We show this by showing (1) $\Phi_1 \le \Phi_0$, and (2) $\Phi_t \le \Phi_{t-1}$ or $\Phi_t \le \Phi_{t-2}$. For simplicity, denote $s_t = i$. Define the difference operator Δ as $\Delta(\cdot) = (\cdot)_{t+1} - (\cdot)_t$. In step t, a vertex x is added to $C_{i,t}$; then one of the four following cases happens. Let k be the number of x's adjacent edges added to $E_{i,t}$, and let h be the number of non-adjacent edges of x added to $E_{i,t}$. We analyze Φ_t in each of the cases.

C1. *When $E_{i,t}$ is full (i.e., the i-th machine is full); x is the first vertex added to $S_{i+1,t+1}$:* Thus $s_{t+1} = s_t + 1$. Counting x and k of its adjacent neighbor vertices, we have $\Delta|S_{i+1}| = k + 1$. The total number of edges allocated is $\Delta|G| = -(k+h)$. However, $\Delta\sigma \le k + h$ as deleting each edge can increase the number of connected components by at most 1. Then we have $\Delta\Phi = \frac{3}{2}\Delta|G| + \frac{1}{2}\Delta\sigma - \Delta s + \Delta|S_{i+1}| \le -\frac{3(k+h)}{2} + \frac{k+h}{2} - 1 + (1+k) = -h \le 0$. (thus $\Phi_1 \le \Phi_0$).

C2. *When $E_{i,t}$ is not full; $x \in S_{i,t} \setminus C_{i,t}$. At the end of step $t+1$, there are still edges in the same connected component as x in G_t not assigned:* Similarly $\Delta|G| = -(k+h)$, $\Delta\sigma_t \le k + h$, and $\Delta|S_{i,t}| = k$. Hence, $\Delta\Phi = \frac{3}{2}\Delta|G| + \frac{1}{2}\Delta\sigma + \Delta|S_i| \le -\frac{3(k+h)}{2} + \frac{k+h}{2} + k = -h \le 0$.

C3. *When $E_{i,t}$ is not full; $x \in S_{i,t} \setminus C_{i,t}$. At the end of step $t+1$, all edges in the same connected component as x in G_t are assigned to $E_{i,t}$:* We have $\Delta|G| = -(h+k)$ and $\Delta|S_i| = k$. By assumption, this step removes fully in G_t the connected component containing x, meaning $\Delta\sigma = -1$. Therefore $\Delta\Phi = \frac{3}{2}\Delta|G| + \frac{1}{2}\Delta\sigma + \Delta|S_i| = -\frac{3(k+h)}{2} - \frac{1}{2} + k \le -\frac{k+1}{2} \le 0$.

C4. *When $E_{i,t}$ is not full but $S_{i,t} \setminus C_{i,t} = \emptyset$. Thus x is a random node chosen in Step 5:* Notice this case happens only when previous step is in Case 3. Similarly $\Delta|G| = -(k+h)$, $\Delta\sigma \le k + h$ and $\Delta|S_i| = k + 1$. Note that Φ will not decrease in this step. However, if take into consideration $\Phi_t - \Phi_{t-1} \le \frac{-(k'+1)}{2}$ for $k' \ge 1$, we have $\Phi_{t+1} - \Phi_{t-1} = (\Phi_{t+1} - \Phi_t) + (\Phi_t - \Phi_{t-1}) \le \left(\frac{-3(k+h)}{2} + \frac{k+h}{2} + 1 + k\right) + \left(\frac{-(k'+1)}{2}\right) \le 0$.

Moreover, right before Alg. 1 terminates, we are in Case 3. Therefore $\Phi_T - \Phi_{T-1} \le \frac{-(k+1)}{2} \le -1$. Furthermore, since $\Phi_{T-1} \le \Phi_0$, we have that $\Phi_T \le \Phi_0 - 1$. In other words $\Phi_0 = \frac{3}{2}|G_0| + \frac{1}{2}\sigma + p \ge \Phi_{\text{end}} + 1 = \sum_{i\in[p]}|S_i| + 1$, which finishes the proof. □

We construct the following example, showing that the analysis for Algorithm 1 (Theorem 4.1) is tight.

Example. In this example, our algorithm is required to partition the edges (in Fig. 3) into four parts. In the worst case, the edge set E_1 can be selected as the green edges, by first including the two green edges in the first line and then the middle edge in the second line. Similarly, E_2, E_3, E_4 may be partitioned as the blue, brown, and

red edges respectively. In this case, $|E| = 12, \sigma = 6, p = 4$, while $|S_{\text{green}}| = |S_{\text{blue}}| = |S_{\text{red}}| = |S_{\text{brown}}| = 6$. Hence $\sum_i |S_i| = 24 = \frac{3}{2} \cdot 12 + \frac{1}{2} \cdot 6 + 4 - 1 = \frac{3}{2}|E| + \frac{1}{2}\sigma + p - 1$, which achieves equality in Theorem 4.1. This example can be easily extended to any p.

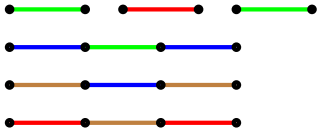

Figure 3: Bound in Theorem 4.1 is tight.

4.2 Upper bound for random power-law graphs

Prior to this work, DBH [17] and HDRF [15] give upper bounds of expected replication factor of their graph edge partitioners for random power-law graphs. In this section, we apply Theorem 4.1 to give an improved numerical upper bound of replication factor for a "typical" random power-law graph, and compare our upper bound with the existing ones in literature.

Power-law graphs specify the degree distribution in the way that the fraction of k-degree vertices p_k is proportional to $k^{-\tau} e^{-k/\kappa}$, where τ and κ are constants. $\tau > 2$ controls the skewness of the distribution, and κ gives a soft upper bound of the vertex degree, i.e., p_k decreases exponentially when $k \geq \kappa$.

To apply Theorem 4.1, we need to estimate the expectation of $|E|/n$ and σ/n. Here we apply the elegant treatment of random power-law graphs by Newman [14] using generating functions, and derive the expected number of connected components and the expected number of edges. Recall that a generating function carries a sequence of numbers as coefficients of a series expansion. The following is a generating function that carries the degree distribution,

$$G_0(x) = \sum_{k=1}^{\infty} p_k x^k,$$

where $p_k = Ck^{-\tau} e^{-k/\kappa}$, with C being a normalization constant so that $G_0(1) = 1$.

Computing $\mathbb{E}[|E|/n]$. We can calculate the expectation of $|E|/n$ as $\mathbb{E}\left[\frac{|E|}{n}\right] = 1/2 \sum_{k \geq 1} k p_k = 1/2 G_0'(1)$.

Computing $\mathbb{E}[\sigma/n]$. We now estimate the term $\mathbb{E}[\sigma/n]$. When $\tau > 2$, it is almost sure that the graph is not fully connected [14]; when $2 < \tau < 3.4785$ and $\kappa \to \infty$, besides small connected components the graph contains a giant component of constant fraction of vertices [1]. To give a quantitative argument, we define the following terms. If we pick an edge and go to one of its end vertices uniformly at random, then the probability $\hat{q}_k$ that the end vertex has degree k is proportional to kp_k. Let $q_k = \hat{q}_{k+1}$ be the probability that the end vertex has outgoing degree k, not counting the edge we come from. This sequence q_k is expressed in the following generating function

$$G_1(x) = \sum_{k=0}^{\infty} q_k x^k = \sum_{k=1}^{\infty} \hat{q}_k x^{k-1} = \frac{\sum_{k=1}^{\infty} k p_k x^{k-1}}{\sum_{k=1}^{\infty} k p_k} = \frac{G_0'(x)}{G_0'(1)},$$

where the last two steps normalize the sequence so that $G_1(1) = 1$. We study the size of a component reached by choosing a random edge and following it to one of its ends. Let $H_1(x)$ be the generating function for the component size distribution, excluding the giant one, i.e., $H_1(x) = a \cdot 0 + \sum t_k x^k$, where a is the probability that this edge leads to the giant component, and t_k is the probability that this edge leads to a small component of size k. Here we follow Newman's assumption [14] that there is no loop in the small components. Hence we have

$$H_1(x) = a \cdot 0 + xq_0 + xq_1 H_1(x) + xq_2 [H_1(x)]^2 + \dots \quad ,$$

and thus

$$H_1(x) = xG_1(H_1(x)). \tag{4}$$

Combining $H_1(1) + a = 1$ and $H_1(1) = G_1(H_1(1))$ we can solve $G_1(1-a) = 1 - a$ for $a \in (0,1)$. Knowing a, we can approximate $H_1(x)$ by solving Equation 4 numerically for the first few hundred terms. Similarly, if we start with a randomly chosen vertex, the size of the small component reached by this vertex is encoded by the generating function

$$H_0(x) = xG_0(H_1(x)) = \sum_{k=2}^{\infty} h_k x^k.$$

The term $\mathbb{E}[\sigma/n]$ can be expressed as

$$\mathbb{E}\left[\frac{\sigma}{n}\right] = \frac{1}{n} + \sum_{k \geq 2} \frac{h_k}{k},$$

where the first term $1/n$ stands for the giant component. As we can numerically compute the first few hundred coefficients of $H_1(x)$, we can numerically compute the coefficients for $H_0(x)$, which are the h_k's.

Expected Upper Bound. Plugging $\mathbb{E}[\sigma/n]$ and $\mathbb{E}[|E|/n]$ into the bound in Theorem 4.1, we obtain an upper bound of expected replication factor for a random power-law graph.

Comparison with Existing Bounds. Let us compare our theoretical bounds with the ones in literature. Prior to ours, DBH [17] and HDRF [15] give upper bounds of expected replication factor for random power-law graphs. In HDRF [15], an average-case analysis

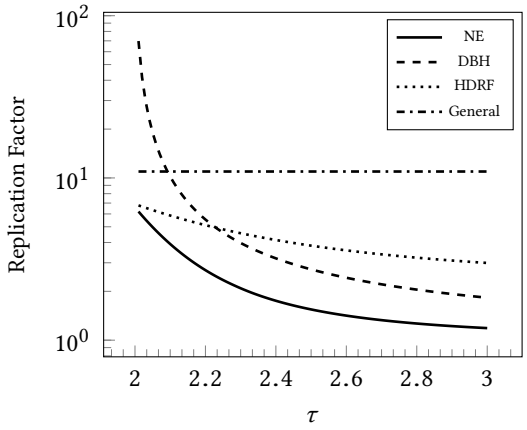

Figure 4: Theoretical bounds of NE, DBH, HDRF and the bound (Lemma 2.2) for general graphs ($p = 30$).

Table 1: Real-world graphs.

Graph	Alias	$\|V\|$	$\|E\|$
com-LiveJournal	LJ	3,997,962	34,681,189
com-Orkut	Orkut	3,072,441	117,185,083
Twitter [9]	TW	41,652,230	1,468,364,884
com-Friendster	FS	65,608,366	1,806,067,135
uk-union [2]	UK	133,633,040	5,507,679,822

is applied to their streaming method to give a bound for power-law graphs. In DBH [17], an upper bound on expected replication factor is derived for their randomized algorithm. However, they only study the case when $\kappa \to \infty$. To apply these bounds, we let $\kappa = \infty$, and consider the graph $n = 10^6$, i.e. set $p_k = 0$ for $k > 10^6$. The results in Fig. 4 indicate that our upper bound is consistently lower than theirs for $\tau \in (2,3)$ with a wide margin. We also plot the general bound derived in Lemma 2.2. As expected, it is worse than the bound we derived in this section that relies on special properties of random power-law graphs.

5 EXPERIMENTAL RESULTS

In this section, we evaluate our *neighbor expansion (NE)* algorithm and compare it against other state-of-the-art partitioners.

To evaluate a partitioner, we consider the following metrics: *workload balance*, *replication factor*, and *time consumption*. For workload balance, we ensure results of each partitioner are 1.1-balanced and do not report the detailed values. We compare the replication factor and running time in Section 5.1. Since one important application of edge-partitioning algorithms is to partition graphs for distributed graph processing systems, we also evaluate whether the given partitioner can reduce the communication cost and execution time on distributed graph processing systems like PowerGraph [6] and PowerLyra [4] in Section 5.2.

Testbed. We evaluate all partitioners and run graph analytic applications (Section 5.2) on a cluster of nine machines, each with 24 Intel E5-2620 2.40 GHz cores and 125 GB RAM connected via Gigabit Ethernet.

Datasets. Five real-world benchmark graphs of various scales are used for our evaluation. Most of the graphs can be found in SNAP [10]. The statistics of the graphs are listed in Table 1.

Competing partitioners. We compare our NE algorithm with six existing edge partitioners, including METIS [8], RAND [6], DBH [17], Oblivious [6], HDRF [15], and Sheep [13]. **METIS** is the state-of-the-art method for vertex partitioning which minimizes edge cut and balances user-defined vertex weight. One can turn a vertex-partitioner into an edge-partitioner while preserving its performance [3]. To transform METIS to an edge-partitioner, we first call METIS with the vertex weight equal to its degree. Then based on the vertex partitioning produced by METIS, each edge is randomly assigned to one of its adjacent vertices' partition. For **Oblivious** and **HDRF**, following [15], we feed the edges in a random order, to balance the resulting partitions.

Table 2: Replication factors for real-world graphs ($p = 30$, $\alpha = 1.1$).

	LJ	Orkut	TW	FS	UK
METIS	2.16	5.24	-	-	-
RAND	8.27	19.48	11.68	11.84	15.99
DBH	5.18	11.97	3.67	6.88	5.14
Oblivious	3.43	6.94	8.60	8.82	2.03
HDRF	3.33	7.27	7.90	8.87	1.62
Sheep	3.33	7.94	2.34	4.45	1.29
NE	**1.55**	**2.48**	**1.88**	**1.98**	**1.04**

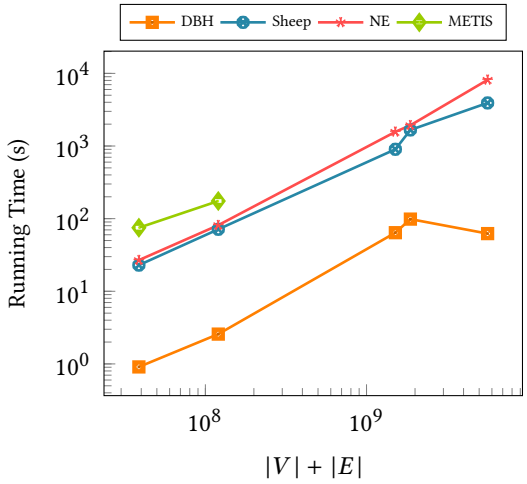

Figure 5: Running time of different partitioning methods.

5.1 Experiments on real-world graphs

In this section, we compare the performance of partitioners on real-world graphs.

Replication factor. As shown in Table 2, our NE algorithm[2] outperforms all existing methods with large margins. Among existing partitioners, METIS gives the lowest replication factor which is consistent with literature [3, 13]. However, METIS runs out of memory when partitioning the Twitter, Friendster and UK-union graphs on our 125 GB RAM machine. This result also echoes the fact reported by [13].

Running time. Fig. 5 plots the running time of all partitioners for a variety of input graphs. As expected, DBH is the fastest since they only scan the input graph once. We did not report the running time of RAND, Oblivious and HDRF since they are integrated in PowerGraph and we cannot get the standalone running time. But their performance should be similar to DBH since they all scan the input graph once sequentially [17]. Using only a single thread, NE and Sheep [13] have similar running time. But Sheep can be speeded up significantly by multi-threading, which is not directly applicable to NE due to the difference in memory access pattern.

[2]Since there is randomness in the NE algorithm, we report the average replication factor, with relative standard error less than 1%.

Table 3: Replication factor, network communication cost (GB), and running time (second) for distributed graph mining.

Task	Graph	PowerGraph + RAND			PowerGraph + Oblivious			PowerLyra + Hybrid_Ginger			PowerGraph + NE		
		RF	Comm.	Time	RF	Comm.	Time	RF	Comm.	Time	RF	Comm.	Time
PageRank	LJ	4.81	40.51	101.07	3.25	26.37	80.77	2.98	12.24	43.12	**1.35**	**3.65**	**39.23**
	Orkut	8.04	54.65	123.37	5.79	37.05	94.67	5.99	25.63	64.90	**1.75**	**6.66**	**44.93**
	TW	6.18	509.96	851.30	2.43	209.46	335.53	2.76	152.25	311.40	**1.38**	**51.45**	**297.97**
	FS	5.39	768.33	1183.90	3.40	411.82	670.10	3.60	294.54	589.24	**1.48**	**92.16**	**392.70**
	UK	6.89	1535.57	2582.93	1.86	540.48	764.10	2.12	386.41	759.26	**1.02**	**55.89**	**554.77**
Triangle Count	LJ	4.81	4.10	3.90	3.25	3.05	3.01	2.99	3.76	3.02	**1.35**	**0.88**	**1.39**
	Orkut	8.04	9.24	9.93	5.79	7.59	8.08	6.02	11.69	10.41	**1.75**	**2.53**	**4.36**
	TW	6.18	91.63	154.73	2.43	56.52	**119.97**	2.74	97.77	188.36	**1.38**	**30.45**	154.91
	FS	5.39	128.12	138.05	3.40	99.22	111.82	3.60	152.69	234.71	**1.48**	**41.29**	**84.18**
	UK	6.89	344.13	2166.17	1.86	157.32	185.54	2.13	299.49	1207.06	**1.02**	**72.34**	**166.07**

5.2 Experiments on graph analytics

Table 3 compares our NE algorithm against two partitioners, RAND and Oblivious, in PowerGraph and one partitioner, Hybrid_Ginger, in PowerLyra, in terms of replication factor, communication cost, and execution time for running the PageRank (100 iterations) and triangle counting applications on a cluster of nine machines. Each result reported in Table 3 is an average of three runs to ensure that their relative standard error is less than 5%. We observe that the communication cost is closely related to the replication factor, roughly as a linear function. This is not surprising because replication factor is designed to model the communication cost. Our NE algorithm optimizes the replication factor and successfully reduces the communication cost for all cases.

Running time depends on replication factor in a similar pattern. For most cases, a reduction of replication factor (hence the communication cost) means a reduction in running time. The only exception is the triangle counting application on Twitter graph. In this case, comparing the NE and the Oblivious methods, we find that a reduction of replication factor dose not reduce the running time. Based on our understanding of the PowerGraph system, this may be explained by several factors. One factor is that, in PowerGraph, the computation and network communication happen concurrently, so the reduction of the communication cost can only reduce the running time when the network communication is the bottleneck of the whole system. Another factor is the potential unbalanced computation workload on each machine. For each vertex with multiple replicas in PowerGraph, one replica is nominated as the master, while others are mirrors. The computation in PowerGraph is conducted on each edge and each master replica. Since our algorithm only guarantees the edge partitioning is balanced, the unbalanced allocation of master replicas may cause congestion on some machines.

In conclusion, compared with other existing methods, our partitioner NE successfully reduces the communication cost for all graphs and reduces the running time for most graphs. In average[3] NE reduces replication factor, communication, and running time by 54%, 66%, and 21%, respectively.

[3]We report the average decrease of ten cases (five from PageRank and five from triangle counting). Each decrease is calculated as $(y-x)/y$, where x is the value of NE and y is the lowest value of existing methods.

6 CONCLUSION AND FUTURE WORK

In this paper, we proposed a new graph edge partitioner Neighbor Expansion (NE) that outperforms other state-of-the-art ones including METIS [8] and Sheep [13] in terms of replication factors. Applying the NE algorithm to distributed graph mining effectively reduces communication cost and running time for applications such as PageRank and triangle counting.

Our algorithm can be further improved in several aspects. Firstly, it will be very interesting if any theoretical approximation results can be proved for the NE algorithm, to explain its good performance in the experiments. Secondly, we might need new data structures to support the NE heuristic in a distributed, multi-thread environment. Thirdly, the NE algorithm needs to load the whole graph edge set in main memory, while other methods like Sheep or DBH only store vertex set. For instance, our implementation of NE takes about 90 GB RAM to partition uk-union [2]. We have made some preliminary attempts to extend the NE algorithm to a streaming algorithm via sampling methods (Appendix B), which is able to partition the clueweb graph [2] ($|V|$ = 978M, $|E|$ = 42.5B) whose edge set exceeds the volume of main memory.

REFERENCES

[1] William Aiello, Fan Chung, and Linyuan Lu. A random graph model for massive graphs. In *STOC*, pages 171–180, 2000.

[2] Paolo Boldi, Massimo Santini, and Sebastiano Vigna. A large time-aware web graph. In *ACM SIGIR Forum*, volume 42, pages 33–38. ACM, 2008.

[3] Florian Bourse, Marc Lelarge, and Milan Vojnovic. Balanced graph edge partition. In *SIGKDD*, pages 1456–1465, 2014.

[4] Rong Chen, Jiaxin Shi, Yanzhe Chen, and Haibo Chen. Powerlyra: Differentiated graph computation and partitioning on skewed graphs. In *Proceedings of the Tenth European Conference on Computer Systems*, page 1. ACM, 2015.

[5] Uriel Feige and Mohammad Mahdian. Finding small balanced separators. In *STOC*, pages 375–384, 2006.

[6] Joseph E Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *OSDI*, pages 17–30, 2012.

[7] 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 *OSDI*, pages 599–613, 2014.

[8] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

[9] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is twitter, a social network or a news media? In *WWW*, pages 591–600, 2010.

[10] Jure Leskovec and Andrej Krevl. SNAP Datasets:Stanford large network dataset collection. 2014.

[11] Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M Hellerstein. Distributed graphlab: a framework for machine learning and data mining in the cloud. *VLDB*, 5(8):716–727, 2012.
[12] 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 *SIGMOD*, pages 135–146, 2010.
[13] Daniel Margo and Margo Seltzer. A scalable distributed graph partitioner. *VLDB*, 8(12):1478–1489, 2015.
[14] Mark EJ Newman, Steven H Strogatz, and Duncan J Watts. Random graphs with arbitrary degree distributions and their applications. *Physical review E*, 64(2):026118, 2001.
[15] Fabio Petroni, Leonardo Querzoni, Khuzaima Daudjee, Shahin Kamali, and Giorgio Iacoboni. Hdrf: Stream-based partitioning for power-law graphs. In *CIKM*, pages 243–252, 2015.
[16] Amitabha Roy, Laurent Bindschaedler, Jasmina Malicevic, and Willy Zwaenepoel. Chaos: Scale-out graph processing from secondary storage. In *SOSP*, pages 410–424, 2015.
[17] Cong Xie, Ling Yan, Wu-Jun Li, and Zhihua Zhang. Distributed power-law graph computing: Theoretical and empirical analysis. In *NIPS*, pages 1673–1681, 2014.

A MIN-RF(p,α) PROBLEM IS NP-HARD

Notation. Given a graph $G=(V,E)$, two vertex sets A,B, let $E(A,B)=\{\{u,v\}: u\in A, v\in B\}$, let $e(A,B)=|E(A,B)|$. Furthermore, we define $E(S)=E(S,S)$ and $e(S)=e(S,S)$.

Theorem A.1. *For any $\alpha\geq 1$, MIN-RF(p,α) problem is NP-hard with respect to n. In particular, we show that MIN-RF$(2,\alpha)$ problem is NP-hard.*

Proof. We claim that MIN-RF$(2,\alpha)$ problem is equivalent to the following MIN-β-separator problem which we will prove to be NP-hard.

Definition 2 (MIN-β-separator problem). *Given a graph $G=(V,E)$, a* β-separator *is a vertex subset S such that $V(G)\setminus S$ can be partitioned into two vertex sets W_1,W_2 such that W_1,W_2 are disconnected from each other and $e(W_i)+e(W_i,S)\leq\lceil\beta e(G)\rceil$ for $i=1,2$. The* MIN-β-separator problem *is to find the smallest β-separator S.*

Lemma A.2. *MIN-RF$(2,\alpha)$ problem is equivalent to MIN-β-separator problem with $\beta=\alpha/2$.*

Proof. Given a graph $G=(V,E)$, let $\{E_1^*,E_2^*\}$ be the optimal solution to the MIN-RF$(2,\alpha)$ problem. Let S^* be the optimal solution to the MIN-β-separator problem with $\beta=\alpha/2$. We show that $|V(E_1^*)|+|V(E_2^*)|=|V|+|S^*|$, and this will infer that two problems are equivalent.

Given E_1^*,E_2^*, let $S=V(E_1^*)\cap V(E_2^*)$. Let $W_i=V(E_i^*)\setminus S$ for $i=1,2$. There is no edge between W_1 and W_2 because E_1^*,E_2^* is a edge partition. And $e(W_i)+e(W_i,S)\leq|E_i^*|\leq\lceil\alpha|E|/2\rceil=\lceil\beta|E|\rceil$, hence S is a β-separator. Therefore, we get $|V(E_1^*)|+|V(E_2^*)|\geq|V|+|S^*|$.

Given S^* which separates $V(G)\setminus S^*$ into two vertex sets W_1^*,W_2^*, we obtain α-balanced 2-edge partition E_1,E_2 as follows. Let $e_i=e(W_i^*)+e(W_i^*,S)$. Then we know $e_i\leq\beta e(G)$. The β-separator gives a natural edge partition for our problem: the first machine contains the vertices $W_1^*\cup S^*$ and the second machine contains the vertices $W_2^*\cup S^*$. The edges in the first machine are $e(W_1^*)\cup e(W_1^*,S^*)$ and x number of edges in $E(S^*)$ with $0\leq x\leq e(S^*)=e(G)-e_1-e_2$. Thus there are e_1+x edges in the first machine. The edges in the second machine are $e(W_2^*)\cup e(W_2^*,S^*)$ and the rest of the edges in $e(S^*)$, thus the number of edges in the second machine is $e_2+e(S^*)-x=e(G)-e_1-x$. We show that we can find an x such that $|E_i|\leq\lceil\alpha|E|/2\rceil$.

WOLG, we assume $e_1\geq e_2$. We separate it into two cases:

(1) If $e_1\geq e(G)/2$, we choose $x=0$. Then the number of edges in the first machine is e_1 and the number of edges in the second machine is $e(G)-e_1\leq e_1$. Because S^* is β-separator, we know $\max\{|E_1|,|E_2|\}\leq e_1\leq\lceil\beta|E|\rceil=\lceil\alpha|E|/2\rceil$.
(2) If $e_1<e(G)/2$, we choose $x=\lfloor e(G)/2\rfloor-e_1$. Thus the number of edges in the first machine is $\lfloor e(G)/2\rfloor$, and the number of edges in the second machine is $\lceil e(G)/2\rceil$. Clearly E_1,E_2 is α-balanced.

Therefore we have obtained an α-balanced 2-edge partition from S^*, which means $|V(E_1^*)|+|V(E_2^*)|\leq|V|+|S^*|$.

Thus we have shown that MIN-RF$(2,\alpha)$ problem is equivalent to MIN-β-separator problem. □

Lemma A.3. *MIN-β-separator problem is NP-hard.*

Proof. The proof is motivated by [5] which proves that a vertex constraint MIN-separator problem is NP-hard. To the best of our knowledge, our work here is the first focusing on the edge constraint of separator and showing that MIN-β-separator problem is NP-hard.

We reduce MAX-CLIQUE problem to MIN-β-separator problem. Given a graph G on n vertices $v_1,\ldots,v_n$, we show that checking whether there is a clique in G containing a fixed v_1 with size k can be reduced to our problem of checking whether there is a separator of size k that satisfies the edge β-separator constraint. (Notice that there are at most n choices of v_1 and at most n choices of k).

We create an auxiliary graph H from G. First create a vertex set W of order $|W|=\max\left(\frac{1-\beta}{\beta}\left(\binom{n}{2}+2e(G)\right)-\frac{3}{\beta}\binom{k}{2},0\right)$. When $k\leq n-1$ as n is large enough, $|W|>0$. Let the vertex set of H be $\{v_1,\ldots,v_n\}\cup\bigcup_{v_i\sim v_j}u_{ij}\cup W$, where u_{ij} are newly created vertices corresponding to each un-ordered pair i,j with v_i,v_j adjacent in G. Thus $|V(H)|=n+e(G)+|W|$. The edges in H are as follows. Within $\{v_1,\ldots,v_n\}$ all edges are connected (thus it is a clique on n vertices). Vertex u_{ij} is only connected to v_i,v_j. There is a complete bipartite graph between W and v_1 while within W it is an independent set. Thus the number of edges in H is $e(H)=|W|+\binom{n}{2}+2e(G)$.

We show that finding the β-separator of size k containing v_1 in H is equivalent to finding a k-clique containing v_1 in G. Suppose $V(H)\setminus S$ can be partitioned into two connected components (each connected components might be disconnected) V_1,V_2. If $v_1\notin S$, then WOLG assume $v_1\in V_1$. Suppose the separator of size k consists of s vertices $U\subseteq\{v_2,\ldots,v_n\}$, x vertices X in W, and y vertices Y in $\{u_{ij}\}$. By the construction, we know that v_1 is connected to all the vertices in $\{v_2,\ldots,v_n\}\setminus U$, and it is connected to all the vertices in $\bigcup_{i,j\text{ not all in }U}\{u_{ij}\}$. Since V_1,V_2 are not connected to each other, all the neighbors of v_1 excluding S should be in V_1. Therefore the following edges are known to be in $E(V_1)\cup E(V_1,S)$: the $|W|-|X|$ edges from v_1 to $W\setminus Y$, the $\binom{n}{2}-\binom{s}{2}$ edges within $\{v_1,v_2,\ldots,v_n\}$ excluding the edges within U, and the edges coming out from u_{ij} with v_i,v_j not both in U and not including the edges coming from vertices in X. The latter case consists of $2(e(G)-e(U))-2y$ edges. Thus we have

$$|E(V_1)\cup E(V_1,S)|\geq|W|-x+\binom{n}{2}-\binom{s}{2}+2(e(G)-e(U))-2y. \quad (5)$$

Since $x+y+s=k$ we know $x+2y \le 2(k-s)$. Also $e(U) \le \binom{|U|}{2} = \binom{s}{2}$. Therefore (5) satisfies

$$\begin{aligned}
&|E(V_1) \cup E(V_1,S)| \\
\ge&|W| + \binom{n}{2} - \binom{s}{2} + 2\left(e(G) - \binom{s}{2}\right) - 2(k-s) \\
\ge&|W| + \binom{n}{2} - \binom{k}{2} + 2\left(e(G) - \binom{k}{2}\right) \qquad (6)\\
=&e(H) - 3\binom{k}{2}. \qquad (7)
\end{aligned}$$

The second to last inequality holds because to minimize the expression over $0 \le s \le k$ we should choose $s=k$ for $k \ge 2$ by simple computation. The last equality holds because $e(H) = |W| + \binom{n}{2} + 2e(G)$.

However, by our choice of $|W|$, we have that $(1-\beta)e(H) = |W| + 3\binom{k}{2}$. Thus $e(H) - 3\binom{k}{2} > \beta e(H)$. However, if this is the case, then (7) tells us that $|E(V_1) \cup E(V_1,S)| > \beta e(H)$. This is a contradiction.

Therefore $v_1 \in S$. Suppose the separator of size k consists of s vertices $U \subseteq \{v_1, v_2, \ldots, v_n\}$ and $v_1 \in U$, x vertices X in W, and y vertices Y in $\{u_{ij}\}$. Suppose $v \in \{v_1, v_2, \ldots, v_n\} \setminus \{v_1\}$. Then similarly, by the construction, we know that v is connected to all the vertices in $\{v_1, v_2, \ldots, v_n\} \setminus U$, and it is connected to all the vertices in $\bigcup_{i,j \text{not all in } U}\{u_{ij}\}$. Since V_1, V_2 are not connected to each other, we know all the neighbors of v excluding S should be in V_1. Therefore the edges we are certain to be in $E(V_1) \cup E(V_1,S)$ include the $\binom{n}{2} - \binom{s}{2}$ edges within $\{v_1, v_2, \ldots, v_n\}$ excluding the edges within U, and the edges coming out from u_{ij} with v_i, v_j not both in U and not including the edges coming from vertices in X. The latter case consists of $2(e(G) - e(U)) - 2y$ edges. Thus we have

$$\begin{aligned}
&|E(V_1) \cup E(V_1,S)| \\
\ge&\binom{n}{2} - \binom{s}{2} + 2(e(G) - e(U)) - 2y \\
\ge&\binom{n}{2} - \binom{s}{2} + 2(e(G) - e(U)) - 2(k-s)
\end{aligned}$$

where the last inequality holds because $x+y+s=k$ and thus $-2y \ge -2(k-s)$. If $1 \le s \le k-1$, we know

$$\begin{aligned}
&|E(V_1) \cup E(V_1,S)| \\
\ge&\binom{n}{2} - \binom{s}{2} + 2(e(G) - e(U)) - 2(k-s) \\
\ge&\binom{n}{2} - \binom{s}{2} + 2\left(e(G) - \binom{s}{2}\right) - 2(k-s) \\
\ge&\binom{n}{2} - \binom{k-1}{2} + 2\left(e(G) - \binom{k-1}{2}\right) - 2,
\end{aligned}$$

where the last inequality is by optimizing the quadratic in s. However,

$$\begin{aligned}
&\binom{n}{2} - \binom{k-1}{2} + 2\left(e(G) - \binom{k-1}{2}\right) - 2 \\
>&\beta e(H) \\
=&\binom{n}{2} - \binom{k}{2} + 2\left(e(G) - \binom{k}{2}\right).
\end{aligned}$$

Thus we have that $s=k$. Thus we must have

$$\begin{aligned}
\beta e(H) =&\binom{n}{2} - \binom{k}{2} + 2\left(e(G) - \binom{k}{2}\right) \qquad (8)\\
\ge&|E(V_1) \cup E(V_1,S)| \qquad (9)\\
\ge&\binom{n}{2} - \binom{k}{2} + 2(e(G) - e(U)). \qquad (10)
\end{aligned}$$

This means that U must be a clique of size k and it contains v_1. Therefore we reduce MAX-CLIQUE problem to MIN-β-separator problem. Since MAX-CLIQUE problem is NP-hard, we have that MIN-β-separator problem is NP-hard. □

The claim is a direct consequence of the two lemmas. □

B THE STREAMING ALGORITHM

Based on Alg. 1, we propose a streaming algorithm (Alg. 3) that considers the trade-off between the partitioning quality and memory consumption. This algorithm only requires $O(|V|)$ memory, hence can process graphs whose edge set exceeds main memory. Our technique is to sample edges uniformly from the original graph, and then run Alg. 1 on the sampled graph.

In total, we need to scan through the edge list[4] from the hard disk twice. In the first scan, we randomly shuffle the edge list, and record the number of vertices, the number of edges, and the degree of each vertex. In the second scan, we stream in the edges one by one from the shuffled edge list, and maintain *CacheSize* number of edges in memory. The value of CacheSize is determined mostly by the memory budget. In general the larger of CacheSize the better the partitioning quality. In practice, we find that CacheSize $= 2\times|V|$ produces balanced partitions of good replication factor.

Like Alg. 1, the streaming algorithm produces edge partition for each machine in turn. In round i, it constructs the working graph consisting of the sampled edges (denoted as $\hat{E}$), and then applies Alg. 1 to allocate a $|\hat{E}|/p-i+1$ fraction of edges for machine i. The allocated edges E_i are immediately stored to the hard disk, while the core set C_i and the boundary set S_i are stored in the main memory for partitioning the unallocated edges.

Prior to round i, the *CheckEdge* procedure (Alg. 4) is used to add extra unallocated edges to the edge set E_j for $|E_j| \le \alpha m/p$ and $j < i$. The reason is that by the time E_j is established, the sampled edge set $\hat{E}$ does not contain the full information of the graph and E_j has fewer edges than expected. The edges allocated to E_j is based on its core set C_j and boundary set S_j. The CheckEdge procedure is applied to the current sampled edge set $\hat{E}$, and also the edges to be loaded into memory. An edge $e=\{x,y\}$ is allocated to E_j if one of the following two conditions is true.

(1) if e is covered by S_j, i.e., $e \subseteq S_j$, or
(2) if e is touched by C_j, i.e., $|e \cap C_j| = 1$, and the degree of each adjacent vertex is less than the average degree of the original graph, i.e., $\max\{d_x, d_y\} \le 2m/n$. (In this case, when edge e is added to E_j the boundary set S_j is augmented as $S_j \leftarrow S_j \cup e$.)

[4]We assume the graph is stored in edge list format. Each line represents an edge as a pair of vertex IDs.

Algorithm 3 Partitioning via sampling.

Randomly shuffle edge list, record n, m, d
$\hat{E} \leftarrow \emptyset$, $E \leftarrow$ shuffled edge list
for $i \in [p]$ **do**
  $\tilde{E} \leftarrow \hat{E}$, $\hat{E} \leftarrow \emptyset$
  for $e \in \tilde{E}$ **do**
    $\hat{E} \leftarrow \hat{E} \cup$ CheckEdge(e, i)
  while $|\hat{E}| \leq$ CacheSize AND $E \neq \emptyset$ **do**
    pick next edge $e \in E$, $E \leftarrow E \setminus \{e\}$
    $\hat{E} \leftarrow \hat{E} \cup$ CheckEdge(e, i)
  $C_i, S_i, E_i \leftarrow$ Expand$(\hat{E}, |\hat{E}|/p-i+1)$
  $\hat{E} \leftarrow \hat{E} \setminus E_i$
  update d, remove degree due to E_i

Algorithm 4 Allocating edges for partitions.

procedure CheckEdge$(e = \{x, y\}, i)$
  for $j \in \{1 \leq t < i : |E_j| \leq \alpha m/p\}$ **do**
    if $e \cap C_j \neq \emptyset$ AND $\max\{d_x, d_y\} \leq 2m/n$ **then**
      $S_j \leftarrow S_j \cup e$
    if $e \subseteq S_j$ **then**
      $E_j \leftarrow E_j \cup \{e\}$
      $d_x \leftarrow d_x - 1$
      $d_y \leftarrow d_y - 1$
      return $\emptyset$
  return $\{e\}$

Memory Consumption. Note that Alg. 3 needs to maintain and work on an edge set of size $|\hat{E}| =$ CacheSize, while the NE algorithm works on the whole edge set of size $|E|$. Both algorithms need to store the core set C_i and boundary set S_i in memory for all the p machines. They also need to maintain the vertex degree d_x in memory. So the total memory consumptions of the NE algorithm and the streaming algorithm are $O(|E| + p|V|)$ and $O(\text{CacheSize} + p|V|)$, respectively. Given that CacheSize $= O(|V|)$ and $p = O(1)$, we show that, by running our NE algorithm on a sampled graph, we can reduce the memory consumption from $O(|E| + |V|)$ to $O(|V|)$. Next, we use the experiments to study the partitioning quality of our streaming algorithm.

Experiments. We compare the replication factor of NE and the streaming algorithm (Alg. 3) in Table 4. To evaluate the scalability of the streaming algorithm we add a new graph *clue-web* [2] of 978,408,098 vertices and 42,574,107,469 edges. Comparing with NE, the streaming algorithm enables us to process much larger graphs with some compromise on replication factor.

Table 4: Replication factors ($p = 30, \alpha = 1.1$)**.**

	LJ	Orkut	TW	FS	UK	clue-web
NE	1.55	2.48	1.88	1.98	1.04	Out of memory
Alg. 3	1.88	4.49	2.83	3.00	1.65	1.94