OffStreamNG: Partial Stream Hybrid Graph Edge Partitioning Based on Neighborhood Expansion and Greedy Heuristic

Tewodros Ayall[1], Hancong Duan[1(✉)], Changhong Liu[1], Fantahun Gereme[2], and Mesay Deleli[3]

[1] School of Computer Science and Engineering,
University of Electronic Science and Technology of China, Chengdu, China
meettedy2123@gmail.com, duanhancong@uestc.edu.cn, 314979677@qq.com
[2] Institute of Fundamental and Frontier Sciences,
University of Electronic Science and Technology of China, Chengdu, China
fantishb@gmail.com
[3] School of Information Science and Engineering,
University of Electronic Science and Technology of China, Chengdu, China
mesay_adinew@yahoo.com

Abstract. Recently, graph edge partitioning has shown better partitioning quality than the vertex graph partitioning for the skewed degree distribution of real-world graph data. Graph edge partitioning can be classified as stream and offline. The stream edge partitioning approach supports a big graph partitioning; however, it has lower partitioning quality, is affected by stream order, and it has taken much time to make partitioning compared with the offline edge partitioning. Conversely, the offline edge partitioning approach has better partitioning quality than stream edge partitioning; however, it does not support big graph partitioning. In this study, we propose partial stream hybrid graph edge partitioning OffStreamNG, which leverages the advantage of both offline and stream edge partitioning approaches by interconnecting via saved partition state layer. The OffStreamNG holds vertex and load states as partition state, while the offline component is partitioning using neighborhood expansion heuristic. And it is transferring this partition state to the online component of Greedy heuristic with minor modification of both algorithms. Experimental results show that OffStreamNG achieves attractive results in terms of replication factor, load balance, and total partitioning time.

Keywords: Edge partitioning · Stream approach · Offline approach · Distributed graph computing · Hybrid edge partitioning · Saved partition state

J. Darmont et al. (Eds.): ADBIS 2020, CCIS 1259, pp. 118–128, 2020.
https://doi.org/10.1007/978-3-030-54623-6_11

1 Introduction

Computing big graph data is nontrivial on a single machine, because of the memory constraint and requires much time to compute the whole input graph. Hence, the best way to process big graphs is using distributed graph processing systems such as Powerlyra [4], Powergraph [5] and Pregel [9]. In all cases, graph partitioning is one of the main component. To computing a big graph in a distributed environment, a graph should be partitioned and distributed into different clusters.

Graph partitioning is a technique to divide a big graph into smaller subgraphs based on different partitioning methods. It is a well-known NP-hard problem [2] to get an optimal solution because it is nontrivial to achieve a minimum cut ratio and maximum load balance. In general, graph partitioning is categorized into two groups, vertex and edge partitioning. Vertex partitioning is also known as edge cut. It divides a big graph into a smaller subgraph by assigning a vertex into the different partition set while considering a minimum edge cut and maximum load balance. These cut edges can act as a bridge to communicate with other partitions. Metis [6], and LDG [12] are some examples of vertex partitioners. Edge partitioning is also known as vertex cut. It divides a big graph into smaller subgraphs by assigning the edge into the different partition set while considering a minimum vertex cut and maximum load balance. These cut vertices can act as a bridge communicator between the partitions. Edge partitioners include Greedy [5], HDRF [11], DBH [13], and NE [15]. The edge partitioners have shown better partitioning quality than vertex partitioners for power-law graph [5], which very few vertices have higher degree, and many vertices have lower degree. Both partitioning methods can further be classified into two as stream and offline approaches.

Stanton and Kliot [12] proposed a stream-based approach for big graph partitioning. The stream-based partitioners ingest vertices or edges as a stream. It applies partitioning decisions on the fly based on partial knowledge of the input graph. The graph data may arrives to the partitioners in Random, Depth First Search (DFS), or Breadth First Search (BFS) order. These arrival orders affect the performance of the stream partitioners [1,3]. Offline partitioners sequentially scan the graph data and store to memory before it makes partitioning.

Stream-based edge partitioners assign a single edge at a time to the partitions based on different techniques. Hashing randomly allocates edges to the partitions based on its hash values. DBH [13] assigns the incoming edges based on the degree information of vertex. It compares the degree of the paired value of edge vertices and gives a hash value of the vertex with a smaller degree to the edge. Greedy partitioning algorithm [5] assigns the incoming edges by checking previously allocated partition state and considering a minimum load balance among each partition. Higher degree replicated first (HDRF) [11] is an edge partitioning algorithm that leverages the advantage of Greedy and adds degree information. It replicates the higher degree first and assigns the incoming edge based on a maximize HDRF computing value. Among stream edge partitioners, Hashing and DBH have a very fast running time; however, they have lower

partitioning quality. On the other hand, Greedy and HDRF have a good partition quality in terms of replication factor and load balance compared with Hashing and DBH; however, they have more running time and are affected by stream order. In general, stream edge partitioners support a big graph partitioning. However, they have lower partitioning quality; require much time to make partitioning and are affected by stream orders compared with offline edge partitioning [15]. NE [15] is an offline edge partitioning and stores all input graph data to memory, then it is iteratively partitioning based on neighborhood relations. It has the best partitioning quality than the stream edge partitioners in terms of replication factor and total partitioning time; however, it does not support a big graph partitioning [15]. In this study, we propose a hybrid graph edge partitioning to improve partitioning quality and reduce the effect of stream order by taking benefits of both stream and offline partitioning approaches via stored partition state. The contributions of this work are as follows:

- We propose partial stream graph edge partitioning OffStreamNG, which uses neighborhood expansion (NE) and Greedy heuristic algorithms for the offline and stream approaches, respectively.
- We introduce the concept of holding and transferring partition state from the offline to stream partitioner with a minor modification of both algorithms.
- We experimentally check the proposed method replication factor, load balance and total partitioning time on real-world graph datasets.

This paper is organized as follows: Section 2 defines graph edge partitioning (vertex cut) problem and Sect. 3 presents the proposed method. Section 4 describes the experimental analysis and results. The conclusion is presented in Sect. 5.

2 The Graph Edge Partitioning (Vertex Cut) Problem

A given undirected graph G defined as $G = (V, E)$, where V is the set of vertices and E is the set of edges, and the size of V and E denoted as $|V| = n_v$ and $|E| = n_e$, respectively. Balanced $p-$way edge graph partitioning problem is defined as, graph G is partitioned into p partitions. Each partition has an edge set $E_k (k \in \{1, 2, ...p\})$. The edge set of each partition is not duplicated, i.e, $E_i \cap E_j = \emptyset$, where $(i, j \in \{1, 2, ...p\}, i \neq j)$.

The graph edge partitioning problem considers two factors: (i) The number of replicas (copy) vertex across partitions are minimized. (ii) The number of edges across the partitions are balanced. Let $P(v)$ be the set of partitions that each vertex $v \in V$ is replicated. Therefore, $|P(v)|$ is size of partitions that stores v. The optimization problem of $p-$way edge partition is defined by Eq. 1.

$$\min_{P} \frac{1}{n_v} \sum_{v \in V} |P(v)| \,. \quad \text{s.t.} \max_{k \in p} |E_k| < \epsilon \frac{n_e}{|p|} \,. \tag{1}$$

where $|E_k|$ and $|p|$ are the size of the edge set of the partition and the number of partitions, respectively. And $\epsilon \geq 1$ is imbalance factor. The performance of

graph edge partitioning can be measured in terms of replication factor (RF), load balance and total partitioning time. Replication factor is an average of vertex replicated in each partition, as given by Eq. 2a. Load balance indicates how fairly edges are distributed in each partition and can be measured by Load relative standard deviation (LRSD), as given by Eq. 2b. Total Partitioning Time (TPT) is the summation of the ingress time (loading time of the input graph) and the running time (the time required for partitioning) of the algorithm.

$$\text{(a) } RF = \frac{1}{n_v} \sum_{i \in p} |P_i(v)|. \quad \text{(b) } LRSD = \frac{\sqrt{(\sum_{k=1}^{p} \frac{|E_k|}{\frac{n_e}{|p|}} - 1)^2 \frac{1}{|p|}}}{\frac{n_e}{|p|}}. \tag{2}$$

3 The Proposed Method

We propose partial stream graph edge partitioning based on neighborhood expansion (NE) and Greedy heuristic with minor modification of both algorithms, and it is called OffStreamNG. The OffStreamNG is the hybrid of NE and Greedy algorithms via stored partition state. Figure 1 shows the architecture of the model. The OffStreamNG model has four sub-components, Modified-NE for offline component, Modified-Greedy for online component, partition state which contains vertex and load states, and input graph splitter. Initially, the input graph is randomly split into two equal parts and is fed into the individual components. While the Modified-NE component is partitioning its input graph data, it is holding the partition state as vertex and load states. On the other hand, the Modified-Greedy component is accepting the other half of the graph data and the partition state as an input to start partitioning. The partition state is continuously accessed and updated by Modified-Greedy to allocate the incoming edges. This partition state is meant to help improve the partitioning quality of the OffStreamNG partitioner.

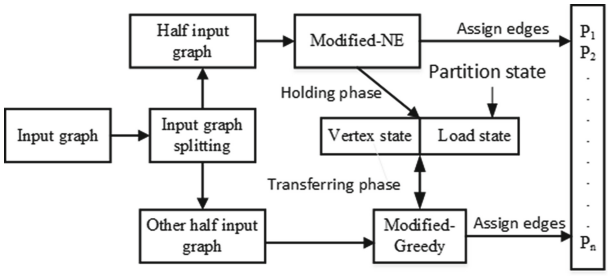


Fig. 1. Architecture of the OffStreamNG model.

3.1 Partition State

A proposed partition state is an intermediate layer of the OffStreamNG partitioner. It is recorded, while the offline component is partitioned its input graph. This partition state is stored in the main memory and accessed by the online partitioner. The partition state gives additional information to the online partitioner to identify appropriate partitions to allocate the incoming edges. The partition state has two states, vertex state, and load state, as depicted in Fig. 2. Figure 2a depicts the vertex state, which holds vertex-ids and partition set (contains all partitions in which a vertex is replicated). Figure 2b shows Load-state, which contains partition-ids and its load balance.

v_1	v_2	v_3	-	-	v_{n-1}	v_n
$\{P_1,P_2\}$	$\{P_1,P_2,P_3\}$	$\{P_1\}$	-	-	$\{P_3,P_{n-1}\}$	$\{P_1,P_n\}$

(a) Vertex state.

P_1	P_2	P_3	-	-	P_{n-1}	P_n
$loadP_1$	$loadP_2$	$loadP_3$	-	-	$loadP_{n-1}$	$loadP_n$

(b) Load state.

Fig. 2. Data structure of partition state: (a) vertex state holds vertex-ids and partition-sets. (b) load state contains partition-ids and its load balance.

3.2 The Offline Component of OffStreamNG Model

The OffStreamNG model has an offline component. This offline component uses Modified-NE and accepts half part of the input graph data. This input graph is partitioned and the partition state is saved by using Modified-NE algorithm. NE [15] is an offline edge partitioning based on neighborhood expansion heuristic. It stores all input graph data to memory and is iteratively partitioning it by growing the core set of vertices via the neighborhood relations. The NE algorithm has got two-component algorithms, edge generation, and edge allocation. In this work, we used the edge generation algorithm as described Algorithm 1 as it is while we have modified the edge allocation algorithm as described in Algorithm 2. The primary purpose of the modification is to enable the algorithm to hold vertex and load states to be used by the online component of our model.

The NE algorithm is iteratively partitioning the graph in p round. In each round k, edge set E_k is selected from the graph. Initially, it is empty edge set. Thus, E_k is expanded in steps until $|E_k| \geq \epsilon \frac{n_e}{p}$. In each round, one vertex y is randomly picked based on neighborhood expansion. The adjacent edges of y is added to E_k and y added to core set C_s. Boundary set $B_s = V(E_k)$, where $V(E_k)$ is the vertex set covered by E_k.

The main objective is to minimize the number of y added into a boundary set based on neighborhood expansion. If $B_s \backslash C_s = \emptyset$ then y is randomly selected from $V \backslash C_s$. Otherwise it is chose based on Eq. 3.

$$y = \underset{v \in B_s \backslash C_s}{\operatorname{argmin}} \mid N(v) \backslash B_s \mid . \tag{3}$$

where $|N(v) \setminus B_s|$ is the number of vertices that will be allocated to the partition k, if y is chose as C_s and its adjacent edges added to E_k.

Algorithm 1. Generate one edge partition E_k.

```
Input: E = E/2, p
Output: E is allocated to p
procedure EXPAND(E, η)                                   ▷ η = ε n_e / p
    C_s, B_s, E_k ← ∅
    while |E_k| ≤ η do
        if B_s \ C_s = ∅ then
            y is randomly selected in V \ C_s
        else
            y ← argmin_{v∈B_s \C_s} | N(v) \B_s |
        end if
        ASSIGNEDGE(C_s, B_s, E_k, y)
    end while
end procedure
```

3.3 The Online Component of OffStreamNG

The offline and online components of our model receive their corresponding graph data portions from the input graph splitter. While the offline component is partitioning its portion, it also is saving vertex and load states which is fed to the online component. We use the Greedy algorithm to build up the online component of our OffStreamNG model with minor modification on it. Greedy [5] is an online edge partitioning algorithm which improves the randomly allocated edges partition based on a heuristic. It is a Greedy sequential heuristic that places the incoming edge to the partitions based on the previously allocated partition state to minimize the expected replication factor.

Let P_{vs} and P_{ls} are vertex state and load state of the partitions, respectively. And minLoad($P_{vs}(V)$) method returns the minimum loaded partition id from the set of $P_{vs}(V)$, where $e = (u, v)|\ u,\ v \in V$. This algorithm assigns the edge e based on the following rules:

Rule 1: If $P_{vs}(u) \cap P_{vs}(v) \neq \emptyset$, then the edge should be allocated to a partition with a minimum load in $P_{vs}(u) \cap P_{vs}(v)$.

Rule 2: If $P_{vs}(u) \cap P_{vs}(v) = \emptyset$ and $P_{vs}(u) \cup P_{vs}(v) \neq \emptyset$, then the edge should be allocated to one of the partition with a minimum load in $P_{vs}(u) \cup P_{vs}(v)$.

Algorithm 2. Modified-Edge Allocation

```
1: P_ls, P_vs ← ∅
2: procedure AssignEdge(C_s, B_s, E_k, y)
3:     C_s ← C_s ∪ {y}, B_s ← B_s ∪ {y}
4:     for a ∈ N(y) \ B_s do
5:         B_s ← B_s ∪ {a}
6:         for b ∈ N(a) ∩ B_s do
7:             E_k = E_k ∪ {e_{b,a}}
8:             E ← E \ E_k                          ▷ Holding partition state
9:             u = e.b, v = e.a
10:            P_vs.addVertexState(u, k)
11:            P_vs.addVertexState(v, k)
12:            if |E_k| > η then
13:                P_ls.addLoadState(|E_k|)
14:                return
15:            end if
16:        end for
17:    end for
18: end procedure
19: procedure getVertexState()
20:    return  P_vs
21: end procedure
22: procedure GetLoadState()
23:    return  P_ls
24: end procedure
```

Rule 3: If only one of the two end edge vertices already has been allocated, then select a partition from the allocated vertex with minimum P_{ls}.

Rule 4: If neither u nor v have been allocated, then the edge is assigned in the partition with the least load of P_{ls}.

However, while the partition state information is very important for the Greedy algorithm to make a decision, it has minimal information at the beginning, which makes the partition quality relatively weak. In this work, we have made a minor modification on the Greedy algorithm. The Modified-Greedy describes in Algorithm refalgo:phasetwo, which takes rich partition state information from Modified-NE algorithm. By getting more partition state information from the offline component, the online component gets sharpened in decision making.

Algorithm 3. Modified-Greedy

Input:$E = E/2,\ p$
Output: Generate a *partition_Id* where E to be allocated.
$P_{vs} \leftarrow GETVERTEXSTATE()$ ▷ Accessing the partition state
$P_{ls} \leftarrow GETLOADSTATE()$
procedure GETPARTITIONID(e, p, P_{vs}, P_{ls})
    $u = e.u, v = e.v$
    if $P_{vs}(u) \cap P_{vs}(v) \neq \emptyset$ **then**
        $partition\_Id = minLoad(P_{vs}(u) \cap P_{vs}(v))$
    else if $P_{vs}(u) \cap P_{vs}(v) = \emptyset$ && $P_{vs}(u) \cup P_{vs}(v) \neq \emptyset$ **then**
        $partition\_Id = minLoad(P_{vs}(u) \cup P_{vs}(v))$
    else if $P_{vs}(u) = \emptyset$ && $P_{vs}(v) \neq \emptyset$ **then**
        $partition\_Id = minLoad(P_{vs}(v))$
    else if $P_{vs}(u) \neq \emptyset$ && $P_{vs}(v) = \emptyset$ **then**
        $partition\_Id = minLoad(P_{vs}(u))$
    else if $P_{vs}(u) = \emptyset$ && $P_{vs}(v) = \emptyset$ **then**
        $partition\_Id = minLoad(P_{ls})$
    end if
    return *partition_Id*
end procedure

4 Experimental Analysis and Results

We implemented OffStreamNG partitioner in an 8 core CPU Ubuntu machine with 64 GB memory. For comparison purpose, we used open-source implementation of edgepart[1] and VGP[2] for NE and stream(Hashing, DBH, Greedy, and HDRF), respectively. We used imbalance factor $\epsilon = 1.1$ and for HDRF $\lambda = 1.1$. We used real-world edge list graph datasets, com-Livejournal from SNAP [8] and Orkut from KONECT [7]. These datasets are randomly ordered. Table 1 shows the characteristics of datasets.

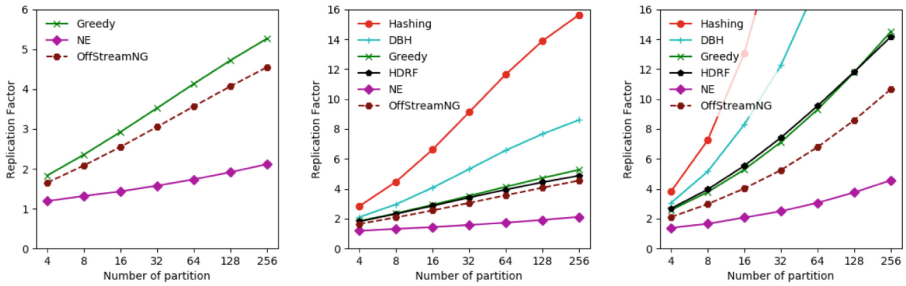
Table 1. Real world graph datasets.

Dataset	n_v	n_e
Com-Livejournal [14]	5,203,764	48,708,948
Orkut [10]	3,072,441	117,184,899

4.1 Experimental Results

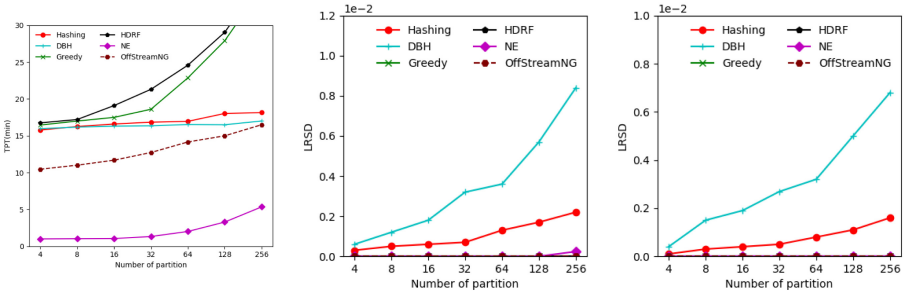
Series of experiments were conducted, and results were carefully recorded. Comparative result analysis is made using the evaluation metrics. Figure 3 shows the replication factor and Fig. 4 shows load balance of com-Livejournal and Orkut datasets.

[1] https://github.com/ansrlab/edgepart.
[2] https://github.com/fabiopetroni/VGP.



(a) Boundedness of RF value. (b) Com-Livejournal. (c) Orkut.

Fig. 3. Replication factor against the number of target partitions (log-log scale) on real-world graph datasets. (a) it shows boundedeness of RF value in Com-Livejournal dataset.



(a) TPT. (b) Com-Livejournal. (c) Orkut.

Fig. 4. Total partitioning time (TPT) and load balance against the number of target partitions (log-log scale). (a) it shows TPT of com-Livejournal. (b) and (c) show load balance.

4.2 Discussion

We evaluate the performance of OffStreamNG by measuring the following metrics:

Replication Factor (RF): RF value is calculated using Eq. 2a and is depicted in Fig. 3. We compared the performance of our OffStreamNG in terms of RF among online edge partitioners such as DBH, Greedy, and HDRF. And also with offline edge partitioner, NE, on real-world graph datasets with a set of target partitions [4, 8, 16, 32, 64, 128, 256]. We calculated average RF from individual RF values on each partition and further averaged these values for all datasets considered. Comparing the calculated average RF values, OffStreamNG performed 62% lower than Hashing, 46% lower than DBH, 20% smaller than Greedy, 18% smaller than HDRF. The RF value showed that OffStreamNG performed far better than the stream edge partitioner. The RF value of OffStreamNG is smaller than other algorithms because it gets more partition state

from the offline component to make a better decision. Generally, Fig. 3a shows that the replication factor (RF) of OffStreamNG bounds between the pure offline and online partitioners.

Load balance: We measured the load balance by LRSD as given by Eq. 2b. The load balance is illustrated in Fig. 4b and Fig. 4c for Com-Livejournal and Orkut, respectively. The curves show that HDRF, Greedy, NE and OffStreamNG performed best as the number of partition grows. Hashing and DBH are the worst performers as load skew grows as the number of target partitions grows.

Total Partitioning Time: We compared OffStreamNG among the stream edge partitioners and the offline NE as shown Fig. 4a on Com-LiveJournal dataset with the number of partitions ranging 4 to 256. The result shows that our OffStreamNG partitioner scored an average TPT improvement of 20% smaller than DBH, 23% smaller than Hashing, 38% smaller than Greedy and 43% smaller than HDRF. Expectedly, NE has smaller TPT than our hybrid partitioners because OffStreamNG is partial streaming. The overall results showed that OffStreamNG scored lower TPT compared with the state of the art stream based partitioners.

5 Conclusion

Graph edge partitioning has dramatically determined the performance of distributed graph processing systems in terms of communication and workload costs. In this study, we proposed partial stream graph edge partitioning OffStreamNG by leveraging both the offline and stream edge partitioning approaches by introducing the concept of holding partition state from the offline and transferring this state to the online partitioner. The OffStreamNG uses neighborhood expansion (NE) and Greedy heuristic for the offline and online components with minor modification of both algorithms, respectively. We compared OffStreamNG with edge partitioners, which OffStreamNG scores diminished value of the replication factor, the optimum load balance, and good total partitioning time.

References

1. Abbas, Z., Kalavri, V., Carbone, P., Vlassov, V.: Streaming graph partitioning: an experimental study. Proc. VLDB Endow. **11**(11), 1590–1603 (2018)
2. Andreev, K., Racke, H.: Balanced graph partitioning. Theory Comput. Syst. **39**(6), 929–939 (2006). https://doi.org/10.1007/s00224-006-1350-7
3. Ayall, T., Duan, H., Liu, C.: Edge property based stream order reduce the performance of stream edge graph partition. J. Phys. Conf. Ser. **1395**, 012010 (2019). IOP Publishing
4. Chen, R., Shi, J., Chen, Y., Zang, B., Guan, H., Chen, H.: PowerLyra: differentiated graph computation and partitioning on skewed graphs. ACM Trans. Parallel Comput. (TOPC) **5**(3), 13 (2019)

5. Gonzalez, J.E., Low, Y., Gu, H., Bickson, D., Guestrin, C.: PowerGraph: distributed graph-parallel computation on natural graphs. In: Presented as part of the 10th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 12), pp. 17–30 (2012)
6. Karypis, G.: METIS: unstructured graph partitioning and sparse matrix ordering system. Technical report (1997)
7. Kunegis, J.: Konect: the koblenz network collection. In: Proceedings of the 22nd International Conference on World Wide Web. pp. 1343–1350. ACM (2013)
8. Leskovec, J., Krevl, A.: SNAP Datasets: Stanford large network dataset collection, June 2014. http://snap.stanford.edu/data
9. Malewicz, G., et al.: Pregel: a system for large-scale graph processing. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, pp. 135–146. ACM (2010)
10. Mislove, A., Marcon, M., Gummadi, K.P., Druschel, P., Bhattacharjee, B.: Measurement and analysis of online social networks. In: Proceedings of the 7th ACM SIGCOMM conference on Internet Measurement, pp. 29–42. ACM (2007)
11. Petroni, F., Querzoni, L., Daudjee, K., Kamali, S., Iacoboni, G.: HDRF: stream-based partitioning for power-law graphs. In: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, pp. 243–252. ACM (2015)
12. Stanton, I., Kliot, G.: Streaming graph partitioning for large distributed graphs. In: Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1222–1230. ACM (2012)
13. Xie, C., Yan, L., Li, W.J., Zhang, Z.: Distributed power-law graph computing: theoretical and empirical analysis. In: Advances in Neural Information Processing Systems, pp. 1673–1681 (2014)
14. Yang, J., Leskovec, J.: Defining and evaluating network communities based on ground-truth. Knowl. Inf. Syst. **42**(1), 181–213 (2013). https://doi.org/10.1007/s10115-013-0693-z
15. Zhang, C., Wei, F., Liu, Q., Tang, Z.G., Li, Z.: Graph edge partitioning via neighborhood heuristic. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 605–614. ACM (2017)