Edge Property Based Stream Order Reduce the Performance of Stream Edge Graph Partition

T Ayall, H Duan and C Liu
School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, China
E-mail: {meettedy2123@gmail.com, duanhancong@uestc.edu.cn, 314979677@qq.com}

Abstract. The graph data is found everywhere in various disciplines, such as in social networks, biological networks, chemical compounds, and computer vision, etc. Currently, the size of graph data has dramatically increased; all disciplines have extracted knowledge from a graph by partitioning and distributing large-scale graph into different clusters using the distributed graph processing system or graph database. However, the graph partitioning has impacted to speed up or slow down the performance of those systems. Even if the stream edge graph partition has shown better partition quality than vertex graph partition for skew degree distribution of a graph and supports big graph partitioning, stream graph edge partitioners are affected by stream orders. In this study, we propose two edge properties based stream order models, TFB(Tree edges First, then Backward edges follow) and BFT(Backward edges First, then Tree edges follow). And study the effect of stream order on stream graph edge partitioners. The results show that TFB and BFT models significantly affect the quality of stream edge partition, except hashing. All algorithms show the best partitioning quality by Random order than other orders, TFB, BFT, BFS(Breadth-First Search), and DFS(Depth-First Search).

1. Introduction
The graphs data naturally is found everywhere in various disciplines such as social networks, biological networks, chemical compounds, transportation, and computer vision, etc. Those disciplines have seen enormous growth information production in the last few years. Recently, big graph analyzing is a nontrivial on a single machine, because of the memory and time constraint to compute the whole graph in a single device. Due to this, the best way to process a big graph is by using distributed computing techniques such as distributed graph processing system Pregel[1] and Powergraph[2] and distributed graph database $G^*$ [3].

Graph partitioning is a technique that divides a big graph into a number of smaller subgraphs based on different partition mechanism. It is well known NP-hard problem [4] to get an optimal solution, because it is nontrivial to achieve a minimum cut ratio and maximum load balance. In general, graph partition can be categorized into two parts such as vertex partition(edge cut) [5, 6, 7, 8, 9] and edge partition(vertex cut) [2, 10, 11, 12, 13, 14].

Both partition methods can be classified into two modes, stream and offline mode. Stream mode ingests vertices or edges as a stream, applying partitioning decisions on the fly based on partial knowledge of the graph. The offline mode sequentially scans and stores the graph data into memory before makes graph partitioning). The vertex partition in stream mode such as [5, 7, 8, 9]. [6] is offline mode. vertex partition has shown the lower quality of partition than
edge partition for skew degree distribution graph such as a power-law graph (a very few vertices have a higher degree, and many vertices have a lower degree) [2]. The edge partition in stream mode such as [2, 10, 11, 12, 13] or an offline mode [14] have shown a good quality of partition for real-world graph datasets.

Fjllstrm [15] studied traditional static vertex partition algorithm, but those unable to partition a big graph. Verma [16] studied an extensive experimental comparison of how to choose a partition technique for distributed graph processing systems and point out three characteristics to select the partition strategies, degree distribution, type and duration of application and cluster size. However, they did not show that the performance of stream edge partitions reduces through stream ordering.

In this study, we propose two stream edge order models and extensively study the performance of stream graph edge partition with stream order. The contribution of this work mention as follows:

- We propose two stream order models, TFB (Tree edges First then Backward edges follow) and BFT (Backward edges First then Tree edges follow).
- We experimentally check, how the quality of the graph edge partition reduces through the stream edge order in various real-world graph datasets.

This paper is organized as follows: we describe background of the study in section 2. Proposed models are presented in section 3. Section 4 describes experimental analysis. Conclusion is summarized under section 5.

2. Background of the Study

2.1. Edge Graph Partition Problem

The PowerGraph[2] proposed $n_{\text{part}}$ edge partition for distributed graph processing system. A given graph $G$ defined as $G = (V, E)$ is a pair of sets where $V$ is the set of vertices and $E$ is the set of edges, and the size of $V$ and $E$ denoted as $|V| = n$ and $|E| = m$, respectively and $E \subseteq V \times V$. We define a partition of a graph $P$ into $n_{\text{part}}$ different set $P_i = (P_1, P_2, P_3, ...P_{n_{\text{part}}})$ such that $P_i \cap P_j = \emptyset$ where $i \neq j$, and $G = P_1 \cup P_2 \cup P_3, ... \cup P_{n_{\text{part}}}$.

The optimization problem of $n_{\text{part}}$ edge partition (vertex cut) defined as.

$$\min_{P} \frac{1}{n}\sum_{v \in V} |P(v)|,$$

s.t. $\max_{P_i} |e \in E|P(e) = p_i| < \epsilon \frac{m}{|n_{\text{part}}|}$.

Let $P(e) \in \{1, 2, ..., n_{\text{part}}\}$ be the partition of edges $e \in E$ is assigned to and $|P(v)|$ is the number of partitions that stores the copy of $v$ called replica, $e$ is single edge and $\epsilon \geq 1$ is imbalance factor.

2.2. Stream Model

A streaming model is a modern approach to partitioning a large-scale graph. It uninterruptedly receives edges or vertices over time and allocated to the partitions as they are being streamed. While reading through the graph edges and vertices, assigned to partitions once and never reassigned. The stream edge can be ordered into three:

(i) Random: The incoming edges data are unpredictable and unknown order.
(ii) Breadth First Search (BFS): It selects one vertex randomly from a connected graph and traverse direct to its neighbors.
(iii) Depth First Search (DFS): It is similar to BFS and selects vertex randomly and traverses to the next neighbor or backtracking.
2.3. Vertex and Edge Partition

The vertex partition is depicted in Fig. 1a. It is one of a graph partitioning approach that divides a big graph into smaller subgraphs by assigning vertices into the different partitions set while considering a minimum edge cut and maximum load balance. These edges can act as a bridge to communicate with other partitions. The edge partition is depicted in Fig. 1b. It is another graph partitioning approach that divides a big graph into smaller subgraphs by assigning the edge to different partitions while considering a minimum vertex cut and maximum load balance. The vertices can act as the bridge to communicate with other partitions.

2.4. Algorithms Description

The stream edge graph partition can be categorized into three ways.

(i) Hash based: It uses hash value to allocate the incoming edges to the partitions. For example Hashing and DBH [13] are categorized as Hash based.

(ii) Constrained based: The incoming edges allocate to the partitions by generate a constrained set. For example Grid [11] and PSD [10] are categorized as constrained based.

(iii) Heuristic based: The incoming edges allocate based on previous partition state information. Greedy[2] and HDRF[12] are categorized as heuristic based.

We use the following notation:- $Ps$ is a set of partition state which contains copy vertex, $du$ is a degree of vertex $u$, $dv$ degree of vertex $v$, $maxsize$ is the size of the partition which have maximum load, $minsize$ is the size of the partition which have minimum load, $npart$ is a number of partitions, $C_{Bal}^{HDRF}$ is compute the score HDRF balance, $C_{Rep}^{HDRF}$ is compute score HDRF replication factor and $C^{HDRF}$ is compute score of HDRF. Hashing assigns the incoming edges to the partitions based on the hash value of the edges as described in algorithm 1. It is very simple to implement and very fast to partition a graph. However, it has a deficient quality of graph partition. DBH assigns the incoming edges based on the degree value of the vertices that form the edges, as shown in algorithm 2. If one of the edge endpoint vertices has a minimum degree, give its hash value of that vertex to the edge. Greedy is a rule-based heuristic, and it has four rules to allocate the incoming edge to the partitions as described algorithm 3. HDRF is an extension of the greedy heuristic that explicitly takes an account vertex degree during placement.

Algorithm 1 Hashing

1: Input:$E$, $npart$
2: Output: Generate a partition id where E to be allocated
3: procedure getPartitionId($e$, $npart$)
4: $partition_{id} = h(e) \mod (npart)$
5: return $partition_{id}$
6: end procedure

edges to the partitions based on the hash value of the edges as described in algorithm 1. It is very simple to implement and very fast to partition a graph. However, it has a deficient quality of graph partition. DBH assigns the incoming edges based on the degree value of the vertices that form the edges, as shown in algorithm 2. If one of the edge endpoint vertices has a minimum degree, give its hash value of that vertex to the edge. Greedy is a rule-based heuristic, and it has four rules to allocate the incoming edge to the partitions as described algorithm 3. HDRF is an extension of the greedy heuristic that explicitly takes an account vertex degree during placement.
Algorithm 2 DBH
1: Input: $E, npart$
2: Output: Generate a partition id where $E$ to be allocated
3: procedure getPartitionId($e, npart$)
4: $u = e.u, v = e.v$
5: $d_u = \text{getDegree}(u)$
6: $d_v = \text{getDegree}(v)$
7: if $d_u < d_v$ then
8: $\text{partition id} = \text{hashValueOf}(u)$
9: else
10: $\text{partition id} = \text{hashValueOf}(v)$
11: end if
12: return $\text{partition id}$
13: end procedure

Algorithm 3 Greedy
1: Input: $E, npart$
2: Output: Generate a partition id where $E$ to be allocated
3: procedure getPartitionId($e, npart$)
4: $u = e.u, v = e.v$
5: if $P_s(u) \cap P_s(v) \neq \emptyset$ then
6: $\text{partition id} = \text{minLoad}(P_s(u) \cap P_s(v))$
7: else if $P_s(u) \cap P_s(v) = \emptyset$ & $P_s(u) \cup P_s(v) \neq \emptyset$ then
8: $\text{partition id} = \text{minLoad}(P_s(u) \cup P_s(v))$
9: else if $P_s(u) = \emptyset$ & $P_s(v) \neq \emptyset$ then
10: $\text{partition id} = \text{minLoad}(P_s(v))$
11: else if $P_s(u) \neq \emptyset$ & $P_s(v) = \emptyset$ then
12: $\text{partition id} = \text{minLoad}(P_s(u))$
13: else if $P_s(u) = \emptyset$ & $P_s(v) = \emptyset$ then
14: $\text{partition id} = \text{minLoad}(npart)$
15: end if
16: return $\text{partition id}$
17: end procedure

decision as shown algorithm 4. The basic idea is a higher degree replicated first and assign an edge to the partition which minimizes compute the score. Grid algorithm is a constrained set of stream edge partition and uses hashing to partitioning edges by arranging the partitions into a square matrix. It makes a constrained set all by taking the row and column as a grid. The incoming edge assigns to the least load of the partition where the intersection of the constraint set found. However, Grid only works for none prime numbers of partitions. PDS algorithm is a constraint-based stream edge partition that computes the constraint set using Perfect Difference Sets. It requires $(p^2 + p + 1)$ number of partitions, where $p$ is a prime number. Its work only for prime number of partitions.

3. The Proposed Models
3.1. Edge Property Based Stream Order
In DFS traversal on a direct graph, the edges can be classified into four type, Tree, Forward, Backward, and Cross Edge. During DFS execute, the classification of edge $(u, v)$ depends on
Algorithm 4 HDRF

1: Input: $E$, $npart$
2: Output: Generate a partition id where $E$ to be allocated
3: procedure getPartitionId($e$, $npart$)
4:     $u = e.u$, $v = e.v$
5:     $d_u = $getDegree($u$)
6:     $d_v = $getDegree($v$)
7:     $\theta(u) = \frac{d_u}{d_u+d_v} = 1 - \theta(v)$
8:     for all partition $i = 1$ to $npart$ do
9:         $C_{Hdrf^{Bal}} = \lambda_{\text{maxsize}} - |i|$
10:        $C_{Hdrf^{Rep}}(u,v,i) = G(u,i) + G(v,i)$
11:        $C_{Hdrf}(u,v,i) = C_{Hdrf^{Rep}}(u,v,i) + C_{Hdrf^{Bal}}(i)$
12:     end for
13:     for all partitions $i = 1$ to $npart$ do
14:         partition $i$ = arg min$_i$ $C_{Hdrf}(u,v,i)$
15:     end for
16:     return partition $i$
17: end procedure
18: procedure $G(v,i)$
19:     if partition $i \in P_s(v)$ then
20:         return $1 + (1 - \theta(v))$
21:     else
22:         return $0$
23:     end if
24: end procedure

whether $v$ visited before or after in DFS traversal. However, undirected graph has only two edges such as Tree edges and Backward edges [17]. In an undirected graph, Tree edges refer to edge visited during DFS traversal and the remain all edges categorize as Backward edges, as depicted in Fig.2. According to the properties of an undirected graph; we propose two stream edge order models to apply for the stream edge partition and analysis the effect of the models on partitioning quality.

Figure 2: The yellow color edges are Tree edges and all other edges are Backward edges.

(i) The Tree edges First then the Backward edges follow (TFB): After all the tree edges of a graph arrive then the Backward edges follow whether in Random, BFS or DFS order as shown in Fig.3a. All the yellow color edges are Tree edges and all the black edges are Backward edges.
(ii) The Backward edges First then Tree edges follow (BFT): After all the Backward edges of a graph arrive whether in Random, BFS or DFS then the Tree edges follow as depicted in Fig.3b.

4. Experimental Analysis and Results

4.1. Experimental Environment

We used Java standalone implementation [12] of each algorithm in a single Ubuntu OS server and 64 GB memory and 8 core CPU.

4.2. Datasets

We used the real world edge list graph datasets[18] to check the effect of the stream order in an graph edge partition algorithms. The characteristics of the datasets see in Table 1. We ordered the datasets Random, DFS, BFS, TFB, and BFT orders. In TFB and BFT order, first the edge of a graph classified as Tree edges and Backward edges. And then edges are ordered based on the principle of the models.

| Datasets      | Number of Vertices | Number of Tree Edges | Number of Backward Edges |
|---------------|-------------------|----------------------|--------------------------|
| roadNet-CA    | 1,965,206         | 1,962,568            | 804,039                  |
| Live journal  | 5,203,764         | 5,199,231            | 43,510,542               |

4.3. Metrics

The quality of graph edge partition evaluates based on the following metrics.

(i) Replication Factors (RF): The total number of copies vertices in each partition divided by the total number of vertices.

\[
RF = \sum_{i=1}^{n_{\text{part}}} \frac{P_i(v)}{n} .
\] (2)

The smallest value of RF means a good quality of graph edge partition.

(ii) Load Balance (LB): Indicate the number of edges in each partition should be at least equal. It can measure using (LSD) and (LRSD).

(a) Load standard deviation (LSD): It is the standard deviation of the number of edge in each partition.

\[
LSD = \sqrt{\left( \sum_{i=1}^{n_{\text{part}}} \frac{|E_i|}{m} - 1 \right)^2 \frac{1}{n_{\text{part}}}} .
\] (3)
(b) Load relative standard deviation (LRSD): It is a relative standard deviation the number of edge in each partition. The value zero indicate equal number of edge in each partitions.

\[ LRSD = \frac{LSD}{m^{part}} . \] (4)

4.4. Experimental Results

We made an experimental analysis on the stream graph edge partitions on various real world graph datasets into \((n_{part} = 31)\) and imbalance factor \((\epsilon = 1.0)\) and the datasets is arranged in all orders. The experimental result of all algorithms are described in Fig.4 and Fig.5.

(a) Replication factor.  

(b) Load balance

Figure 4: The replication factor and load balance in roadNet-CA dataset.

(a) Replication factor.  

(b) Load balance

Figure 5: The Replication Factor and Load balance in Livejournal dataset.

4.5. Discussion

We measured the RF and load balance value of sparse graph roadNet-CA as shown in Fig.4 and dense graph Livejournal as shown Fig.5 in the stream edge partition algorithms with different orders. The result showed that all algorithms increased or decreased replication factor and also increased or decreased the load balance by all order except hashing. Our models showed comparable effect as BFS and DFS order in sparse graph as well as in dense graph in all algorithms. In general, the quality of edge graph partition best in random order than other orders, because the quality of edge partition determined by RF and LRSD value. A among those two factor one of it increased or decreased; reduced the performance of the edge partition.
Even if the HDRF is affected by stream order, it has the best replication factor and load balance among the stream edge partition.

5. Conclusion

In this study, property based stream orders, TFB and BFT are presented. And the effect of stream orders, Random, DFS, BFS, the proposed model TFB, and BFT on stream-based graph edge partition algorithms are analyzed with dense and sparse real-world graph datasets. We have demonstrated that the proposed stream order and other stream orders significantly reduced the quality of graph edge partitioners except hashing. Even if graph edge partitioners has the best partitioning quality in Random order, some edges come in a predefined order like TFB, and BFT also significantly reduced the quality of stream edge graph partitioning. Still, it is an open research direction to bring an edge partition algorithm that tolerates stream order, the minimum replication factor, and the maximum load balance while supporting a big graph partition.

References

[1] Malewicz G, Austern MH, Bik AJ, Dehnert JC, Horn I, Leiser N, Czajkowski G. Pregel: a system for large-scale graph processing. In Proceedings of the 2010 ACM SIGMOD International Conference on Management of data 2010 Jun 6 (pp. 135-146). ACM.

[2] Gonzalez JE, Low Y, Gu H, Bickson D, Guerstrin 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) 2012 (pp. 17-30).

[3] Labouseur AG, Birnbaum J, Olsen PW, Spillane SR, Vijayan J, Hwang JH, Han WS. The G* graph database: efficiently managing large distributed dynamic graphs. Distributed and Parallel Databases. 2015 Dec 1;33(4):479-514.

[4] Andreev K, Racke H. Balanced graph partitioning. Theory of Computing Systems. 2006 Nov 1;39(6):929-39.

[5] Echbarthi G, Kheddouci H. Fractional greedy and partial restreaming partitioning: New methods for massive graph partitioning. In 2014 IEEE International Conference on Big Data (Big Data) 2014 Oct 27 (pp. 25-32). IEEE.

[6] Karypis G. METIS: Unstructured graph partitioning and sparse matrix ordering system. Technical report. 1997.

[7] Nishimura J, Ugander J. Restreaming graph partitioning: simple versatile algorithms for advanced balancing. In Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining 2013 Aug 11 (pp. 1106-1114). ACM.

[8] 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 2012 Aug 12 (pp. 1222-1230). ACM.

[9] Tsourakakis C, Gkantsidis C, Radunovic B, Vojnovic M. Fennel: Streaming graph partitioning for massive scale graphs. In Proceedings of the 7th ACM international conference on Web search and data mining 2014 Feb 24 (pp. 333-342). ACM.

[10] Halberstam H, Laxton RR. Perfect difference sets. Glasgow Mathematical Journal. 1964 Jul;6(4):177-84.

[11] Jain N, Liao G, Willke TL. Graphbuilder: scalable graph etl framework. In First International Workshop on Graph Data Management Experiences and Systems 2013 Jun 23 (p. 4). ACM.

[12] Petrini F, Querzoni L, Davdjee K, Kamali S, Iacoboni G. Hhrf: Stream-based partitioning for power-law graphs. In Proceedings of the 24th ACM International on Conference on Information and Knowledge Management 2015 Oct 17 (pp. 243-252). ACM.

[13] Xie C, Yan L, Li WJ, Zhang Z. Distributed power-law graph computing: Theoretical and empirical analysis. In Advances in Neural Information Processing Systems 2014 (pp. 1673-1681).

[14] Zhang C, Wei F, Liu Q, Tang ZG, Li Z. Graph edge partitioning via neighborhood heuristic. In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining 2017 Aug 13 (pp. 605-614). ACM.

[15] Fjllstrm PO. Algorithms for graph partitioning: A survey. Linkping: Linkping University Electronic Press; 1998 Sep.

[16] Verma S, Leslie LM, Shin Y, Gupta I. An experimental comparison of partitioning strategies in distributed graph processing. In Proceedings of the VLDB Endowment. 2017 Jan 1;10(5):493-504.

[17] Cormen TH, Leiserson CE, Rivest RL, Stein C. Introduction to algorithms. MIT press; 2009 Jul 31.

[18] Leskovec J, Krevl A. SNAP Datasets: Stanford large network dataset collection.