Turkish Journal of Computer and Mathematics Education Vol.12 No.6 (2021), 5335-5350

Research Article

Stream-Based Vertex cut partitioning with Buffer support for Power-law graphs(SVBP)

N. Mithili Devi[a], Dr. Sandhya Rani Kasireddy,[b]

[a] Research Scholar, Department of Computer Science,Sri PadmavathiMahilaViswavidyalayam,Tirupathi,Andhra Pradesh,India
[b] The Director,IIIT-RK Valley, RGUKT-AP, Andhra Pradesh, India
[a]mithilisreedhar@gmail.com

Article History: Received: 10 November 2020; Revised 12 January 2021 Accepted: 27 January 2021; Published online: 5 April 2021

Abstract: Evolution of online technologies and usage resulted in massive data collection, analyzing and visualizing such huge data through graphs has become one of the interesting areas of research. The Big graphs created for massive data are very huge and cannot be processed by a simple machine at an adequate amount of time any-more. So the Big graphs are to be partitioned and stored on different machines to process and analyze them quickly. Traditional graph partitioning methods can no longer does this task since it follows of-fline processing and requires to store and access the entire graph from one machine leading to memory bottlenecks and also time consuming. Hence, Streaming Graph partitioning methods have gained momentum and these methods can partition real time online graphs directly and efficiently. Streaming graph partitioning methods takes stream of edges along with its end vertices as input into a Scheduler machine. The scheduler machine intern partitions the graph and assigns the nodes and edges to different partitions as re-ceived. Since entire graph cannot be made available to the scheduler machine at any given point of time, it assigns edges to partition ma-chines based on using some partition criteria that may not be optimal. Scheduler's decision can be notably improved if partitioning is done only after receiving sufficient information of the node or edge being allocated. This paper recommends an efficient Buffer-based edge streaming algorithm called SVBP for graph partitioning. This method implements the idea of delaying the assignment of few edges and restream them at right time to improve partitioning effi-ciency. Our method uses a Buffer to store the edges whose partition-ing is delayed. The SVBP algorithm is evaluated on real-time power-law graphs that are notably large. Our method is able do the job of partitioning efficiently on all the graphs by keeping the replication factor minimum and balancing the load across partitions legitimately good compared to other algorithms.

Keywords: Graph Partitioning, Streaming, Restreaming,Vertex cut, Edge-cut, Streaming Graph Paritioning, Buffer-based

1. Introduction

In this big data era, Machine learning and Data mining with versatile massive datasets have dragged attention from researchers and business people. In particular, recent years have seen the advent of bulky real-time graphs in areas like Social networking (Twitter and Facebook), Road and Telecommunication networks etc.,[3][4] and this kind of Large-scale graphs cannot be controlled and processed by one machine because of memory bottlenecks and time constraints. Hence, abundant research works happened in this regard resulting revolutionary changes in handling large-scale graphs. Rather than struggling to handle large graphs as a whole [3], people started to divide and distribute them onto several partition machines before processing. Subsequently Graph partitioning attracted many researchers interest and is most craved area of study in Big data domain.

Graph partitioning come under NP-Complete problem and majority of naturally created graphs exhibit power-law degree distribution. Traditional graph partitioning methods are offline and needs whole graph to be kept before partitioning that incurs lot of memory bottlenecks and time delays[2]. Streaming graph partitioning methods are grabbing attention now a days as these methods accept flow of edges and nodes as input from online graphs and does distributed processing and segregating. If the parts of graph is distributed among several machines then computation cost will be high in general. Hence, the aim of Streaming Graph partitioning is to diminish the communication rate and also maintain load balancing in every partition nearly equal. The streaming graph partitioning method is called efficient if communication cost is low and Load balancing is nearly equal in all partitions[1]. The two frequently used steaming partitioning methods are edge cut partitioning and vertex cut partitioning as explained in Figure 1. Edge cut partitioning receives flow of graph vertices as input and divides by cutting the edges and assigning the vertices to separate partitions. The ambition of Edge cut partitioning is reduce the cross-partition edges and maintain the load balancing across the partitions. Contradictorily, Vertex cut partitioning receives flow of edges as input and divides by cutting the vertices (replicating the vertices) and assigning the edges to separate partitions. The ambition of vertex cut partitioning is to diminish the count of replications of vertices (Replication Factor) and maintain balanced load across all partitions [10]. Majority of real-time graphs display power-law mark scattering, that means majority of vertices have relatively few neighbors and a very few vertices have more count of neighbors [5][6]. This paper emphases on Vertex cut partitioning methodology due of the point that in case of Power-law distributed graphs, Vertex cut partitioning models are serving better compared to Edge cut partitioning. Actually majority of the available methods like HDRF, DBH, S-Power Graph etc., are single-pass algorithms and produces balanced partitions to a fair extent [4][5][19]. But

the fruitfulness of dividing can be further enhanced by partitioning edges only when sufficient neighboring node information is supplied to the scheduler [7]. This can be accomplished by keeping such edges whose neighboring data is inadequate to partition in a Buffer and re streaming them after specified time limit. S-Power Graph algorithm is one-pass algorithm designed for natural skewed power-graphs that produces balanced partitions with minimum vertex cut. S-Power Graph is an enhanced version of adapted Power Graph algorithm. S-Power Graph algorithm generally tries to soothe the partitions and it may assign edges to inappropriate partition as it cannot guess the pattern of future graph.

This paper suggests a Stream-Based Vertex cut partitioning with Buffer support for Power-law graphs (SVBP) algorithm to obtain enhanced partitioning ability and to minimize Replication Factor while keeping load balance approximately equal on all partitions. SVBP method accomplishes this by delaying the partitioning and placing the edge whose neighboring data is lacking in a Buffer, where buffer space is adjusted as per the convenience of Scheduler's memory availability. Also, the edges kept in Buffer window are re streamed into the Scheduler machine for partitioning after specified time limit. The edge assignment will be delayed whenever enough of neighborhood vertices are not processed till then.

The main contributions of this paper are briefly outlined as follows:

- Better Load Balancing among partitioning machines.
- Reducing superfluous vertex replication by delaying the edge assignment if enough of its neighbor vertices have not yet visited.
- Introducing Buffer window to keep unallocated edges whose assignment decision is postponed.
- Better Partitioning is attained by holding the edge in Buffer until enough of information is received.

The paper is structured as follows. Our reading on the related work and the current advancements in streaming graph partitioning is expressed in section II, Problem formulation of Vertex cut streaming graph partitioning, Methodology and assumptions are deliberated in section III, Our recommonded SVBP architecture, Algorithm and methodology to add and delete edges of buffer is deliberated in section IV. The dataset, evaluation setup and Investigational results are analyzed in Section V. Lastly, Conclusion is provided in Section VI.

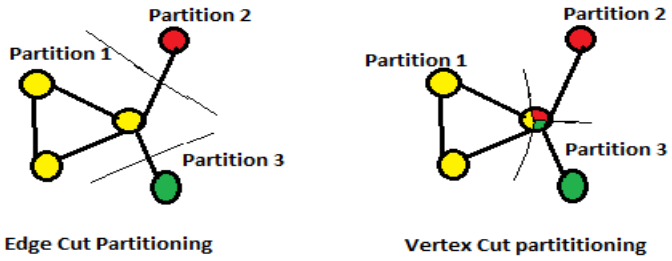

Fig. 1. Partitioning the graph into three parts using Edge cut and Vertex cut Partitioning

2. Related Work

This section is all about our study related existing research on Streaming graph partitioning. During the past few years, online streaming Big graph partitioning attracted many researchers interest [2]. Immense growth of big graphs accelerate the advent of streaming graph partitioning. The very first streaming graph partitioning method was developed by Stanton and Kliot in the year 2012. This method introduced simultaneous receiving and partitioning the graph.Considering edge cut vs. vertex cut streaming algorithms, we review two groups of streaming graph partitioning algorithms namely: edge cut and vertex cut partitioning [8].In general majority of real-time comprehensive graphs exhibit power-law degree dissemination by nature and many researchers established the point that vertex cut partitioning algorithms perform better than edge cut partitioning algorithms in case of power-law graphs.

Edge cut Partitioning

In online (streaming) edge cut partitioning algorithms, vertices arrive in sequence and the algorithms assign them to different partitions [7]. Majority of streaming algorithms are single-pass algorithms and they prohibit partition refinement after assignments. Spatio-Temporal Interaction Networks and Graphs Extensible Representation (STINGER) provides a basis for analyzing structured graph data and facilitates experimenting and advancements in the area of big data and big graphs by providing a dynamic graph data structure. Stinger facilitates addition and deletion of edges from scale-free graphs and quick query processing[14]. Planted Partition Model is another edge-partitioning method recommonded by Tsouraksakis to improve partion quality by using complex stretch walks for partitioning. To minimize inter partition communication in low complexity structures

and to accommodate faster query processing, Wang and Chiu developed one more mountable streaming partitioning system.

Ja-be-ja is another distributed vertex swapping method developed by Rahiman et al. This method is built based on local search and reduces inter partition communication and is appropriate for sparse network[11]. Leopard is another light weight edge cut partitioning method recommonded for streaming and restreaming vertices from a dynamic graph[23]. This method facilitates addition and deletion of partitions also.

Stanton et al. studied different heuristics, and noticed the best enactment with the linear deterministic greedy (LDG) heuristic. This model assigns vertex with relatively more neighbors to one common partition, and the algorithm weights the assignment by a penalty function built on the dimension of the partitions. FENNEL is another online streaming edge cut partitioning algorithm intended for maximizing the modularity[9]. Fennel's basic concept is to interpolate between maximizing the co-location of neighboring vertices and minimizing that of non-neighbors. WStream is another Window-based one-pass streaming edge cut partitioning algorithm designed for large-scale graphs that minimizes communication cost across partitions and balances load among partitions[17].

Vertex cut Partitioning

In online streaming vertex cut graph partitioning algorithms, edges arrive in sequence and the algorithms distributes those to partition machines. Majority of these algorithm are one-pass algorithm and they prohibit partition refinement after assignments. Greedy-heuristic and DBH are innovative streaming vertex cut graph partitioning algorithms. HDRF, S-Power Graph are two partitioning algorithms developed subsequently for proficient partitioning of Power-law graphs by with improvised load balancing and minimized vertex replication. However, these methodologies are centralized and don't scale neither horizontally nor vertically[12][13]. Following HDRF and S-Power Graph, IOGP and CLDA takes benefit of both HDRF and greedy methods. CLDA uses HDRF technique for high degree vertices and applies greedy technique for low degree vertices. IOGP uses edge cut algorithm for low degree vertices and vertex cut algorithm for high degree vertices. In addition, HoVerCut[4], RBSEP, ADWISE are popular efficient edge partitioning methodologies developed recently[18]. A mountable streaming multi threaded graph partitioning algorithm Hovercut provides both vertical and horizontal scalabity with buffering technique to stake incoming edges. When it comes to dynamic graphs Hovercut doesn't suit and bring out efficiency deprivation over period of time[2]. SGVCut is another vertex cut partitioning method for specially developed for random walks foundation that tries to minimize replication factor and inter partition communication to the level possible[17]. ADWISE is the window-based streaming partitioning method specially designed to increase partitioning quality and reduce partitioning time by adjusting window capacity according to processing speed of the partitioning[16].

This paper recommends a Stream-Based Vertex cut partitioning with Buffer supportalgorithm; edges are streamed to the scheduler machine. The scheduler verifies if the end vertices of the processing edge has enough neighbors to take decision of assigning, if no sufficient neighbors then all such edges will be kept in Buffer window. The edges kept in Buffer window gets examined for sufficient neighbors periodically and allocated to Partition machine accordingly. The edges must be retained in Buffer only for specified time limit. If time limit exceeds then such edges will be restreamed to the Scheduler machine to allocate them to emptiest partition machine. There by the edge assignment can be delayed until sufficient neighbors gets streamed ad allocated to en route for better partitioning. Our model is designed apparently that edge assignment is done at right time to acquire improved replication factor and load balancing. The edges are getting streamed and re streamed multiple times before partitioning with buffer support. Therefore this technique achieves significant betterment in reducing vertex cut since the choice to allocate an edge has major effect on its upcoming coupled edge from the buffer. As per our observations and study, this model is completely new and not used anywhere.

Table. 1. The notation used in the paper

Symbol	**Description**
G	Input graph
E	collection of Edges of G
V	collection of vertices of G
k	Total count of partition machines considered
λ	count of edges to be handled by scheduler before restreaming edges from buffer

MAXV	Maximum time limit that edges can reside in Buffer
δ(v)	degree of vertex v
S(v)	collection of partitions in which vertex v is replicated
P(k)	Current collection of edges be held by k^{th} partition
P_{ID}	ID^{th} partition is denoted by P_{ID}
C	Buffer capacity

3. Problem Formulation

In general majority of natural real- life graphs display power-law degree distribution. Which means maximum of the vertices have comparatively few neighbors, whereas a very little vertices have a plenty of neighbors and the probability of a vertex having d is demarcated as

$P(d) \propto d^{-\alpha}$

where α denotes a small positive constant that controls the degree.

Vertex cut Partition

The algorithm explained here follows vertex cut partitioning strategy[5]. Suppose we have Graph named G = (E,V) where V indicates collection of vertices and E indicates collection of edges, then k-way vertex cut partitioning of the Graph G into collection of k partitions $P = (p_1, p_2, \ldots, p_k)$ is done so that specific vertices G is replicated in multiple partitions denoted by set S(V) and $S(v) \subseteq P$. Finally, the allotment of vertices is done apparently that the count of vertices replicated (Replication Factor) is minimum and all existing the partitions gets allocated with approximately equal load of edges(Load balancing).

Min $\frac{1}{|V|}\sum_{v\in V}|\boldsymbol{S}(\boldsymbol{v})|$ such that $\max_{p\in P}|\boldsymbol{E}(\boldsymbol{p})|<\boldsymbol{\Omega}\frac{|E|}{|P|}$

Where $\Omega \geq 1$ denotes small positive constant to specify the method acceptance to load balance. Note that here in streaming graph partitioning, continuous flow of edges are passed as input data to the scheduler for partitioning.

Edge Streaming Methodology

We appraise a streaming partitioning framework in which edges arrive in a stream and each edge appears only one time in the stream[5][19]. The edges arrive along with end vertices. Moreover, every time a vertex arrives together with few of its edges. For SVBP, the order that the vertex arrives is most predominant since it can influence the enactment of the methods significantly. In our streaming setting, the vertices arrive in a definite order with all of their out edges arriving in a probable order. The vertex stream is received by scheduler in the below specified three different orders that are measured as a standard from existing works:

- **Random (Rnd) order:** Here the vertices arrive into the scheduler in a random order specified by a random permutation of the vertices.
- **Breadth First search (BFS) order:** A starting vertex is selected from the graph and new vertices arrive if they are found by a breadth first search.
- **Depth First search (DFS) order:** DFS works in a same manner like BFS ordering except that depth-first search will be used.

Data Processing Model and Assumptions

We consider that graph edges are read online in streaming fashion. It is presumed that, we cannot access the whole graph at once, and therefore, we cannot perform global operations. Additionally, since we do the partitioning job along with reading online graph, the degree of individual vertex is not identified in advance. The degree supply is disclosed gradually when the graph edges arrive in stream. Therefore we can only have access to subgroup of edges besides its aggregate values that are streamed till then. Replication factor(RF) may be identified as the overall count of vertices replicated divided by the total count of vertices[20][21]. This RF will be minimized by cutting more high degree vertices. Reason for this is that when the degrees of vertices are under the power-law distribution, there will be numerous vertices with small degree and a comparatively few vertices are with high degree. So if choices are available, we always choose to cut a vertex with relatively higher degree.

The recommended method is an improvisation of S-Power graph algorithm by adding buffering mechanism with an inkling of delaying partitioning decision. To produce well-organized buffering strategy, the buffer dimension is adjusted in harmony with scheduler's memory space. This is being done due to inadequate buffer

space. The scheduler machine receives flow of edges as input and assigns them either to some partition machine or keep them in buffer relying on the obtainability of edge information and end vertices data. This mechanism is depicted in Fig. 2.

4. SVBP Architecture

Our system has four major components namely Scheduler Machine, Partition Machine, Buffer Window and Vertex table as illustrated in Figure 2. We used an Edge Stream Generator that receives edge stream from online graph and send to Scheduler for partitioning. The Scheduler contains our partitioning algorithm SVBP that considers edges one after the another from the flow of input edges supplied, processes the edge and assign to available partition machines. Vertex table contains already allocated vertices data together with their degrees and allocated partition details of S(V) pertaining to edges that are already allocated. This Vertex table gets updated each time the edge is allotted to partition machine. This information is needed for allocation of future edges that arrive as input stream. Figure 2 illustrates the structure of recommonded model.

• **Scheduler Machine**: Scheduler receives stream of edges as input from the taken from the online graph. Scheduler considers one edge at a time and does the job of partitioning. Scheduler makes use of Vertex table information while placing the edge in desired partition machine. Suppose e is the edge considered with end vertices V_i and V_j. Consider $S(V_i)$ and $S(V_j)$ values from Vertex table. S(Vi) indicates collection of partition machines inside vertex V_i is replicated and $S(V_j)$ indicates collection of partition machines inside vertex V_j is replicated. If both S(Vi)=Ø and S(Vj)=Ø then the edge is moved to Buffer by delaying the choice of assigning that edge. Otherwise edge may be allotted to partition P_{ID} calculated using Algorithm 1.

• **Partition Machine:** Each partition Machine receives allocated edge from the Scheduler machine and stores. Partition machines intern communicate among themselves to sustain the existing parameter values.

. Vertex Table: Vertex table Contains info related to allocated vertices. Soon an edge is allotted to Partition machine, its end vertices details gets updated in the Vertex table. Vertex table stores partial degrees of assigned vertices and set S(V) in which vertex V is replicated. Vertex table gets updated after every edge partitioning.

.Buffer Window: Buffer window contains collection of edges whose assignment is postponed. Whenever sufficient neighboring data is deficient for edge assignment, those edges must be kept waiting in Buffer window until either sufficient information is obtained or it reaches to MAXV waiting time.

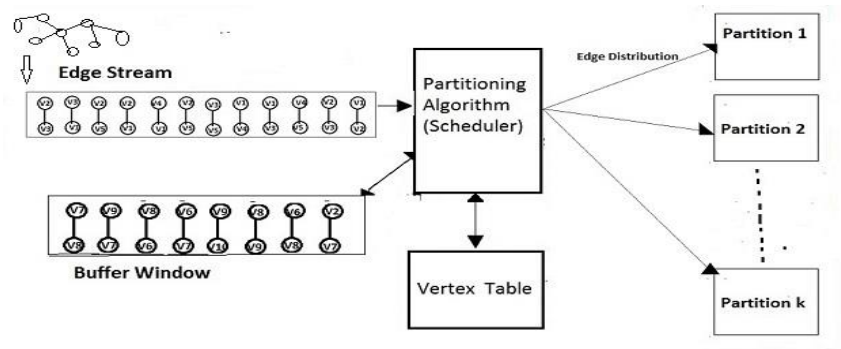

Fig. 2. SVBP Architecture

The recommended model makes use of S-Power graph algorithm. As formerly mentioned S-Power graph is one-pass algorithm that has a good enactment on skewed power-law graphs. S-Power graph algorithm is an enriched version of Greedy algorithm on Power graphs, it prioritizes cutting those vertices that have the highest degree[15]. However, unlike Greedy, S-Power graph considers Score value for partitioning. These models can stream the edges only once and the edge assignment made once cannot be modified. Moreover the edge streaming and assigning is done depending on partial neighboring information obtained till then. Even if sufficient info is not obtainable, these models are compelled to assign edges to partition that can never be changed. Since the graph is processed online, the unseen part of the graph is revealed in future. Therefore the enactment of the partitioning model can be improvised by delaying the assignment process of the edge till enough of neighboring information is gained and restreaming all such edges by retaining them in Buffer window. The edges of the Buffer can be restreamed into the scheduler multiple times until its time limit gets expired. The assignment decision is taken when sufficient neighboring info is available or time limit gets expired. This way our algorithm improvises the partitioning process as illustrated in Fig.2.

The Scheduler machine receives stream of edges as input and considers edge by edge, makes use of Alorithm1 and Algorithm 2 to calculate ID value. Edge e is distributed to the partition P_{ID}. We believe in delaying the edge

allocation when sufficient neighboring data is lacking, this could lead to an improvised partitioning. Our regular partitioning algorithms like DBH, HDRF, S-Power graph, Greedy etc., forcibly chooses to allocate the edge to some existing partition due to which the replication factor increases[20]. However, the partitioning mechanism can be improvised by delaying allocation process to future until the maximum percentage of the graph has been noticed. In this paper, we recommend an improvisation to S-Power Graph algorithm by adding buffer window and multiple streaming mechanism that helps to delay the edge allocation decision. The buffer window size is adjusted based on scheduler's buffering capacity and hence efficiency is improved.

Methodology for adding edge into buffer window for restreaming

Soon the edge enters into the scheduler machine, the Algorithm retrieves information related to end vertices of that edge from vertex table associated with Scheduler machine. Suppose e is the edge considered with end vertices V_i and $V_j$and the neighboring information collected from Vertex table may fall into any of the following three cases.

Case1: Neither of the edge vertices V_i and V_j are assigned till then. That means $S(V_i) \cup S(V_j)=\emptyset$

Case 2: Either of the edge vertices are already assigned to some partition machine. That means $S(Vi) \cup S(Vj) \neq \emptyset$

Case 3: The edge vertices have some common partitions allocated. That means $S(Vi) \cap S(Vj) \neq \emptyset$.

Case 2 and Case 3 works similar to S-Power Graph and edge e gets allocated to partition P_{ID} as per the processing done by Algorithm 1. Case 1 leads to sending edge to Buffer by delaying assignment process since sufficient neighboring information of end vertices is at hand for partitioning at this instant. Hence, it is preferable to delay the partitioning decision and reconsider at later stage as discussed in Algorithm 2.

Methodology for deleting an edge from Buffer to restream:

As discussed in section 4.1, all the edges whose assignment decision got delayed to are kept in Buffer window. It is always wise to delay the edge allocation whenever enough of neighboring data is lacking and assign later so that better partition can be attained. The edges kept in Buffer are reconsidered for streaming after streaming $\boldsymbol{\lambda}$ edges by the scheduler. After processing every $\boldsymbol{\lambda}$ edges, the scheduler takes edge from buffer and allocates based on conditions specified in Algorithm 2. The edges kept in Buffer are considered by Scheduler machine for restreaming periodically after processing every $\boldsymbol{\lambda}$ number if edges from online stream. When the edge taken from Buffer is restreamed into Scheduler machine, it will again check if enough of neighboring information is present, if so it assigns to partition machine otherwise the edge must be retained back into buffer for future streaming. Like this the edge kept restreamed for multiple times until the time limit MAXV is reached. The Buffer can retain edges only for time limit of MAXV after that the scheduler is forced to allocate it to barest partition as explained in Algorithm 2. There must be a counter maintained with each edge deposited in Buffer that gets updated periodically. Whenever the counter exceeds MAXV value then the Scheduler is compelled to assign it to emptiest partition even though sufficient neighboring info is not attained. The whole process of deleting an edge kept in buffer can be illustrated with two cases as given below:

Case 1: After processing $\boldsymbol{\lambda}$ edges suppose the Scheduler wants to reexamine edge kept in buffer then it will use Algorithm 1 that checks if any of end vertices are by now allocated(i.e. $S(v_i) \neq \emptyset$ or $S(v_j) \neq \emptyset$), if so the edge will be partitioned to P_{ID} estimated based on Algorithm 1.

Case 2: During restreaming, if the end vertices have not yer placed in some partition and the edge cannot be retained in buffer anymore (i.e. $S(vi) = \emptyset$ and $S(vj) = \emptyset$), at that moment the edge must be assigned to the barest partition decided by Algorithm 2.

The SVBP Algorithm

Here we present SVBP, a Buffer based algorithm tailored for skewed power-law graphs. Our partitioning scheme tries to place each strongly connected component with low-degree vertices into same partition by cutting high-degree vertices and replicating them on several partitions. This assists to diminish communication cost among partitioning machines by reducing count of vertex replications. Though, procuring degrees of allotted vertices in an online streaming fashion is difficult our algorithm achieves best results because whenever sufficient neighboring information is not procured, the partitioning decision gets delayed. Since the complete graph is not admitted formerly and edges are getting streamed from online graph as input, a Vertex table with partial degrees of the vertices and collection of partitions where all that allocated vertices gets replicated ie., S(V) can be maintained. The Vertex table is continuously updated while input is streamed and assessed. Whenever Scheduler machine chooses to allot an edge to partition machine, the vertex table gets updated with the matching end vertices degree values and allocated partition sets. Sometimes the scheduler cannot take optimum assignment due

to lack of neighboring information of edge's end vertices then it delays the decision of assigning to future so that better partitioning is attained. All such vertices must be kept in Buffer for restreaming.

More formally, when processing edge e ∈ E connecting vertices v_i and v_j, the SVBP algorithm retrieves their partial degrees and increments them by one.

In this paper, we observe two partitioning algorithms. We define each of them as follows.

Let P(k) denote the current collection of edges pertaining to the k^{th} partition,

and let [p] = {1…. p}. We define

maxedges = $\max_{k\in[p]}\{|P(k)|\}$ and minedges = $\min_{k\in[p]}\{|P(k)|\}$ (1)

We refer to a certain partition by its index ID. Then the ID^{th} partition is denoted by P_{ID}.

The two partitioning algorithms we considered are **Random** and **Balance** works as below:.

Random: The random method simply assigns each edge via a random hash function as implemented in PowerGraph. So it is the most naive algorithm. The edge e is allocated to P_{ID} where ID is decided by:

ID = hash(V_L) (2)

hash(V_L) here is a randomized hash function.

Where V_L = argmin D(w) , where w∈ $\{Vi, Vj\}$

Balance: A new constraint called Balance is added to avoid imbalance. This modified version plays a major role as baseline which demonstrates the enactment of the Power Graph partitioning instead of greedy algorithms, when allocating an edge, each algorithm will first compute a Score for each partition. It is the partition with the maximum Score that the edge will finally be allotted to.

Score(k) = 1{if k∈ S(Vi) otherwise 0} + 1{k∈S(vj) otherwise 0} + balance(k);
(3)

Where k∈[p] and

balance(k) = $\frac{maxedges-|p(k)|}{maxedges-minedges+1}$ (4)

where P(k) denote the current collection of edges belong to the k^{th} partition

if $\frac{\max_i|P(i)|}{avg\,|P(i)|}$ ≥1.1,then the edge e is allocated to P_{ID} here ID is computed by:

ID = argmax {balance(k)} (5)

k∈ $[p]$

otherwise the edge e is allocated to partition P_{ID} here ID is computed by:

ID = argmax {Score(k)} (6)

k∈ $[p]$

Algorithm1: Buffered_Stream

Input: v_i, v_j, k

Output: partition p

1. get collection of partitions S(V_i) where v_i is reflected from vertex table

2. get collection of partitions S(V_j) where v_j is reflected from vertex table

3. If S(V_i)=Ø and S(v_j)= Ø then

Insert edge in to Buffer by delaying the partitioning and wait for next edges.

else

4. Calculate maxedges and minedges using equation (1)

5. Calculate Balance value using equation (4)

6. Calculate Score value using equation (3)

7. Find the highest Score value ID using equation (5) or (6)

8. Assign the edge e to the partition P_{ID}

9. Update degree of V_i and degree of V_j in vertex table

10. Add Partition p into $S(V_i)$ and $S(V_j)$ in vertex table

11. End if

Algorithm2: Remove_Buffer

Input: V_i, V_j, counter

Output: partition p

1: if count>=**λ** then

2. Retrieve $S(V_i)$ and $S(V_j)$ from vertex table

3. If $S(V_i) \neq \emptyset$ or $S(V_j) \neq \emptyset$ then

4. Call Buffer_stream(V_i,V_j,k)

5. Else

6. Find emptiest partition P_{ID} from $S(V_i) \cup S(V_j)$

7. if(Count_V_iV_j> MAXV * **λ**) then

8. Assign edge to the barest partition P_{ID}

9. Else

10. Keep edge e in buffer

11 End if

12.Else

13. Keep edge e in buffer

14.End if

15. Repeat steps 1 to 14 and update count_V_iV_j at regular intervals.

5. Experimental Evaluation

In this segment we valuate and relate the enactment of our SVBP method with HDRF, DBH, Greedy-heuristic and S-Power graph methods. With a view to achieve this we implemented the suggested algorithm together with other algorithms of specified methods in Java and evaluated their efficiency on real massive graphs. In this segment, we first give a sketch of evaluation settings like datasets used, evaluation methodology followed. Also, we endorse the efficiency metrics used and lastly, we talk about the obtained results.

5.1 Datasets

The graph datasets used for our trials are considered from snap website (Stanford large network data collection (SNAP), http://snap.stanford.edu/data. All the considered graphs are unweighted and undirected. Since all considered graphs are real-life graphs, all graphs follows power-law distribution. Table 2 exhibits the details pertaining graph datasets used for our efficiency evaluation.

Table 2. Characteristics of Datasets

Name of Dataset	**Count of vertices**	**Count of edges**	**Type**
facebook	22,470	1,71,020	Social Network

Amazon	3,34,863	925872	Product Network
Enron	36,692	183,831	Communication Network
CondMat	23,133	93,497	Collaboration Network
Brightkite	58,228	2,14,078	Location based online social network
Github	37700	289003	Social network
AstroPh	18,772	198,110	Collaboration network
Eather-Deezer	28,281	92752	Social network

5.2 Efficiency Metrics

The Efficiency metrics considered for evaluating our model is provided below:

i) Replication Factor
ii) Load Balancing factor and
iii) Partitioning Efficiency..

Replication Factor

Replication factor measures the count of vertices replicated among multiple partitions. It is accustomed to approximate the communication cost across multiple partitions. Replication factor may be stated as the aggregate count of vertices replicated divided by total count of vertices. The replication factor will be minimum if we replicate less count of vertices. As per power-law distribution, the real-life graphs we considered contains fewer count of vertices with higher degree and larger count of vertices with smaller degree[11][13]. Therefore better replication factor is attained by replicating vertices with higher degree rather than replicating vertices with smaller degree. So only in our graph partitioning algorithms we always try to choose to cut a vertex with relatively higher degree.

Finally, the replication factor indicates how many vertex copies an edge partitioning algorithm creates. We calculate it as follows:

$$\text{RF} = \frac{Total\ Vertex\ copies}{Total\ number\ of\ vertices}$$

The mathematical form of replication factor is can be represented as below:

$$\text{RF} = \frac{1}{|v|}\sum_{v \in V} |S(v)|$$

Where, |V| indicates total count of vertices of a graph and S(v) indicates collection of partitions in which vertex V gets replicated. The Partitioning quality is said to be good if Replication factor value is approximately nearing to 1. In our evaluation process we calculated Replication factor for all the real-life graphs that are considered.

Load Balancing Factor

Load Balancing factor is yet another measure for evaluating the graph partitioning manner. Load balancing indicates how well the edges load is distributed across multiple partitions. The algorithm aims to attain balanced load on all partitions. Load balance factor is calculated with the formula given below.

$$\max_{p \in P} E(p) < \Gamma \frac{|E|}{|P|}$$

where |E| indicates total count of edges and |P| indicates total count of partition machines across which the graph edges get distributed. E(p) indicates total count of edges allocated to partition p and Γ indicates Load balancing factor that displays the degree of satisfactory deviation from the balance. The load is considered to be well balance ifΓ is small. The above inequality illustrates the acceptable count of vertices that can be allotted to a partition to attain better load balancing[14]. The above inequality can be rewritten as below to calculate Load balance factor.

$$\Gamma = \max_{p \in P} E(p) * \frac{|P|}{|E|}$$

Partitioning Efficiency

Partitioning Efficiency is measured using Time complexity and Space complexity analysis. Time complexity is calculated by counting execution time from the begin of partitioning process till the end. Since our SVBP algorithm receives online edge stream as input, Time needed for input to arrive is moreover necessary. In general edges gets processed only once and few edges whose assignment decision is delayed gets moved to buffer. Worst case analysis is considered if more count of edges decision gets delayed or if edges take little extra time to arrive at the scheduler. But trials illustrated that better partitioning is carried out by delaying the allocation process. So Time complexity of our algorithm is O (n + m* **λ**) + O (p* C *s log k), here n indicates count of edges streamed, m indicates count of edges whose decision is delayed and hence kept in buffer and p indicates count of restreams happened. Likewise **λ** is count of edges streamed to scheduler before processing an edge deposited in buffer, k indicates count of partitions and s indicates the partition count C indicates buffer capacity.

Space Complexity is nothing but amount of space occupied by our algorithm to complete execution[9][16]. The space complexity of our algorithm is almost same as S-Power graphs algorithm. Although buffer one added feature to our algorithm still its space complexity is same as HDRF since the space needed by buffer is adjusted based on scheduler capacity. Space engaged by Vertex table is same as HDRF algorithms Vertex table.

5.3 Evaluation Scenario

The productivity of our SVBP algorithm is evaluated on different datasets specified in Table2, results are analyzed and matched with HDRF, DBH, Greedy technique and S-Power Graph methodologies in section 5.4. The results evaluation analysis is done for the parameters specified in section 5.2. Unlimited streaming model is used to stream the input of graph edges into the scheduler. Various possible combinations of partitions with varieties of datasets are used in our evaluation process. The trials are conducted with varied window sizes, varied partitions and varied datasets. Different Buffer window sizes considered are 100, 200, 300,400,500, 600, 700 and 800. Different count of partition machines used are 2,4,8 and 16. Different datasets used are mentioned in table 2. All these datasets are considered from snap. The results are valuated by changing count of partitions and Buffer capacity.

Experimental results

The evaluated experimental results presented as follows: Productivity of SVBP algorithm is assessed concerning to replication factor, Load balancing and Execution time. We first evaluate the effect ofreplication factor on different real world datasets by considering 8 partition machines conferred in section 5.4.1. We then evaluate recommonded method on different graph datasets with varied Buffer sizes and varied count of partition machines deliberated in section 5.4.2. We then evaluated load balance of SVBP on diverse datasets for Buffer size 16 deliberated in section 5.4.3. Effect of MAXV on replication factor is deliberated in section 5.4.4. Productivity of our SVBP algorithm for different input dimensions is conversed in section 5.4.5. Effect of different input stream order on replication factor is conferred in section 5.4.6.

Throughout our evaluation process we considered the value of as 5 and Buffer capacity as 100, 200, 400

5.4.1 Effect of Replication Factor on different Real world datasets

We have executed all five algorithms namely SVBP, HDRF, DBH, Greedy and S- Power graph on different datasets considered from snap by keeping count of partition machines as 8. The replication factor calculated for all five algorithms are compared and the results are validated in Figure 3. Our SVBP algorithm produced better replication factor over HDRF, DBH, Greedy and S- Power graph.

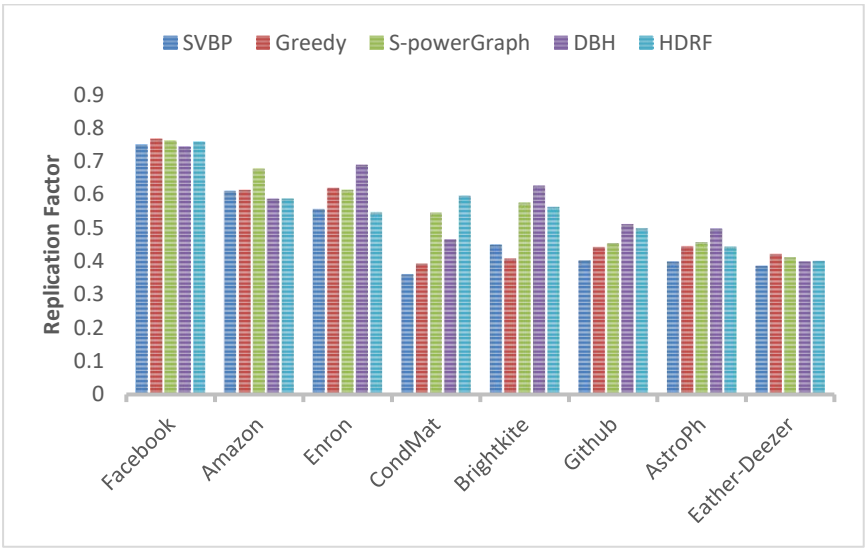

Fig. 3. Effect of Replication Factor on different Real world datasets

Effect of different Buffer size on different datasets

Buffer is used to maintain all those edges whose partitioning is delayed. We executed our SVBP algorithm on deliberated datasets separately with varied buffer sizes. The replication factor generated for each buffer size on a specific dataset is noticed and the results are illustrated in Figures 4a, 4b,4c,4d,4e,4f,4g and 4h.The SVBP algorithm intended to utilize the Buffer space to provide efficient partitioning efficiency by delaying partitioning decision. The results of Figure 4 demonstrates the consequence of different Buffer dimensions on the splitting efficiency of SVBP algorithm. It is anticipated that as the buffer size increases the partitioning efficiency also becomes better that generates better Replication factor. Because if the Buffer size is larger then more count of edges partitioning can be delayed and it leads to better partitioning. Smaller the replication factor, more efficient the algorithm becomes Anyway, as communicated in Figure 4, in some exceptional cases all algorithms resulted in a higher Replication Factor for the bigger Buffer volumes than the lesser ones. Balance parameter is calculated while partitioning. To maintain the partitions balanced our algorithm defines another parameter called score which plays major role in deciding partition machine to which the edge gets allocated. The scheduler always assigns the edge to a partition with thoroughgoing Balance or Score values. Figure 4(a) illustrates that the SVBP algorithm performs better on Facebook dataset for 4 partitions and 16 partitions in diminishing replication factor. Figure 4(b) demonstrates effect of replication factor on Amazon dataset for varied buffer sizes with varied count of partitions. We notice that the algorithm gave better results for count of partitions 4 leading to better efficiency. Figure 4(c) demonstrates the SVBP algorithm out performed on Enron dataset with Buffer size 700 for all partition counts. Figure 4(d) demonstrates the effect ofreplication factor on the Brightkite dataset with a different count of Buffer window sizes and different partitions. We notice that for having count of partition machines as 4 our algorithm achieves better outcome except for Buffer size 400 and Buffer size 700, which obtains a somewhat higher Replication Factor, otherwise the Replication Factor declines the Buffer size escalates. As showcased in Figure 4(f) and 4(g) for datasets Condomat and AstroPh our SVBP algorithm outperformed in sinking the Replication factor since the Buffer size increases. We notice that, for Facebook dataset, partition count 16 out performs other partition settings and Condmat datasets outperforms l for the partition count 2.

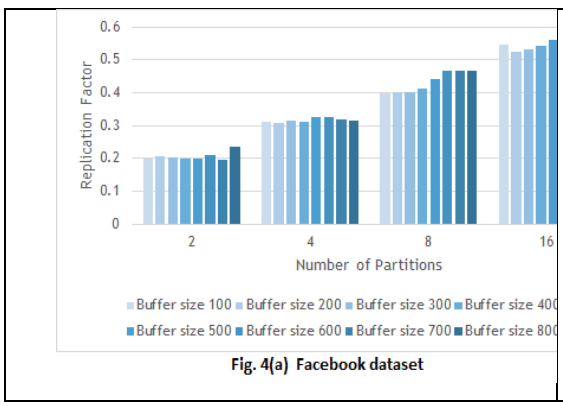

Fig. 4(a) Facebook dataset

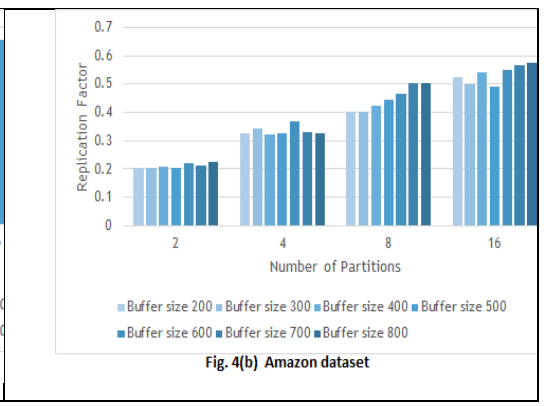

Fig. 4(b) Amazon dataset

Fig. 4 Effect of different Buffer size on different datasets

5.4.3 Effect of Load Balancing

Load balancing means all partitions must get loaded with nearly equal count of edges. Load balancing is another powerful factor that decides the productivity of partitioning algorithm. If the partitioning model is capable to distribute the edges to all partition machines equally then the load is declared well balanced and the technique is called an efficient one. To evaluate the influence of load balancing on dissimilar datasets, we executed our SVBP algorithm with all datasets and correlated the results with HDRF, DBH, Greedy and S-Power graph. The outcomes are talked about in Figure 5. The load balancing factor of our SVBP algorithm remains nearly the same with all datasets. If load balancing factor is 1 then the load is declared well balanced among partition machines. All the methods maintained load balance around 1. Our SVBP algorithm maintained almost same load balancing on all methods due to implementation of buffer window and delaying the partitioning whenever the edge doesn't have sufficient neighboring information.

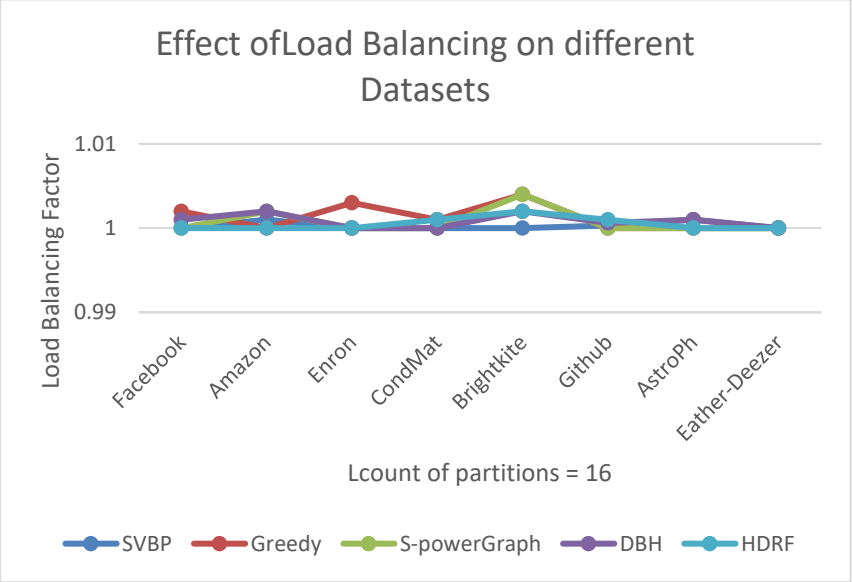

Fig. 5. Influence of Load balancing on dissimilar datasets

5.4.4 The influence of the MAXV on replication factor

MAXV is the time limit defined to know how long the edge is retained in Buffer. This parameter controls the act of recommended algorithm. The influence of MAXV on replication factor is deliberated in this segment by executing the algorithm on Facebook dataset. The end result is revealed in Figure 6. The edge whose end vertices doesn't get replicated in any of $S(V_i)$ or $S(V_j)$ will be kept in buffer for MAXV time. The edge gets restreamed into the scheduler periodically after every **λ** count of edges gets processed. The edge can spend MAXV time in buffer and once this time exceeds and still it doesn't have sufficient neighboring information then the edge should get allocated to emptiest partition. From the graph demonstrated in Figure 6, it is understood that the replication factor declines as the parameter MAXV increases. So better partitioning can be attained if keep the edge for long time in buffer.

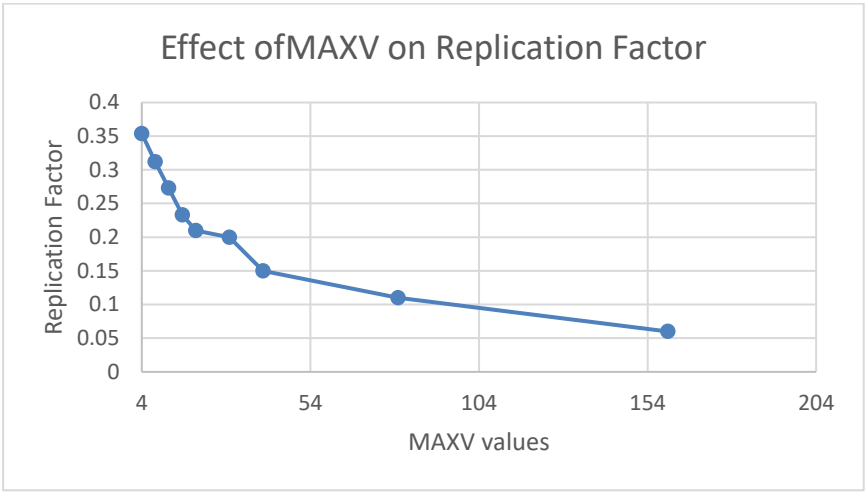

Fig. 6 The influence of the MAXV on replication factor

5.4.5 Efficiency of SVBP for various input volumes

Effectiveness of our algorithm for various input volumes is evaluated and the outcomes are demonstrated in Figure 7. Our algorithm is capable to reach relatively smaller replication factor even for larger volumes of datasets. As dipicted in figure, larger the input data, larger the replication factor also. The replication factor got increased according to input size for all different datasets nearly in similar manner. In fact as the input volume increases, the algorithm receives more neighboring information subsequently better partitioning. Therefore as input volume increases, the buffer gets more count of edges whose assignment decision is delayed.

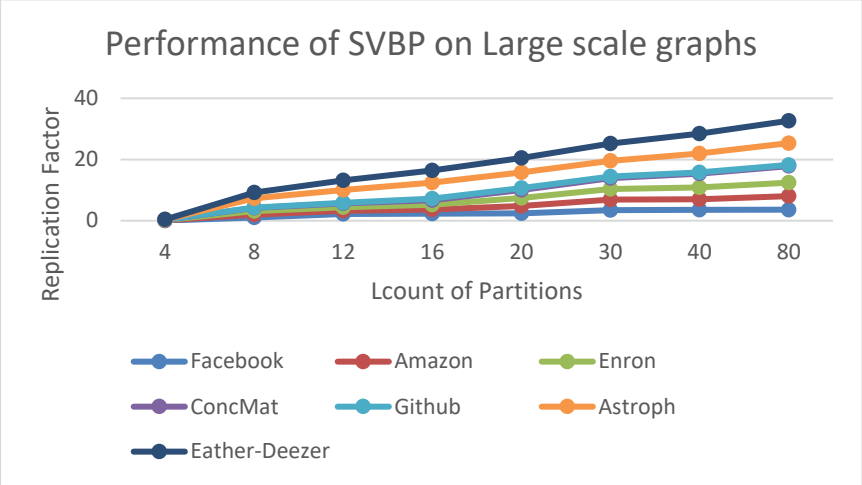

Fig. 7 Efficiency of SVBP for various input volumes

5.4.6 Influence of Input stream order on Replication factor

In Figure 8, we compared the replication factor on different real-world graphs in different orders by SVBP algorithm. Observing the graph we can state that out of three input stream orders Depth First search(DFS) order, Breadth First Search(DFS) order and Random order, Replication factor also increased along with input size. Out of all three orders, the BFS order outperformed the other two.

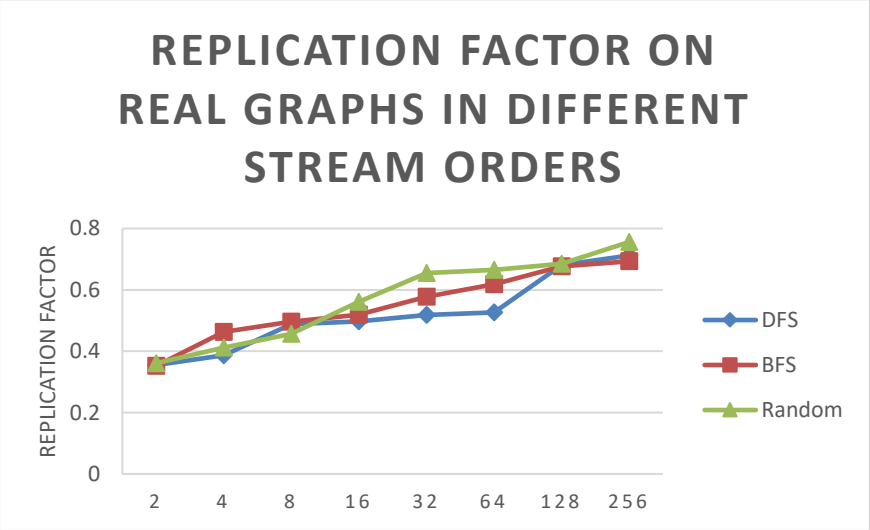

Figure 8 Influence of Input stream order on Replication factor

6. Conclusion

In this paper we have studied streaming graph partitioning by passing Edge stream. Specifically, we have recommendeda innovative streaming graph partitioning method for Real graphs by adopting vertex cut strategy and called it SVBP. Sometimes the partitioning decisions made by the existing streaming graph partitioning algorithms are not appropriate as they make partitioning only from the existing part of the graph that is received till then and the edges gets streamed in continuously. Therefore the partitioning can be improvised with our algorithm. Instead of allocating the edge in one-pass there and then it is streamed, it would be nice to delay the allocation of certain edges whose neighboring information is insufficient to make decision. All such edges whose assignment is delayed gets moved into a buffer window maintained. The edges kept waiting in buffer until sufficient neighboring information is arrived and gets restreamed into scheduler periodically multiple times till it gets allotted to a partition. The buffer volume gets adjusted as per scheduler's memory space. As far as we know this is the first work to methodically study the effectiveness of vertex cut in streaming graph partitioning. Trials on big scale graphs prove that our method is more appropriate for partitioning skewed natural graphs than the previous approaches with satisfactory load balance and replication factors. Our method helps to partition either the raw graph data streamed online or graph data loaded from a storage device. In our upcoming work, we would love to pursue more variants of the suggested strategy theoretically and scientifically.

References

1. Zainab Abbas, VasilikiKalavri, Paris Carbone, and Vladimir Vlassov. Streaming Graph Partitioning: An Experimental Study. PVLDB, 11(11): 1590-1603, 2018.
2. PeiroSajjad, H., Payberah, A H., Rahimian, F., Vlassov, V., Haridi, S. (2016)
3. Boosting Vertex-Cut Partitioning For Streaming Graphs. IEEE In-ternational Congress on Big Data (BigData Congress 2016)
4. Loc Hoang, RoshanDathathri, Gurbinder Gill, KeshavPingali . CuSP: A Customizable Streaming Edge Partitioner for Distributed Graph Analytics. May 2019.
5. IEEE International Parallel and Distributed Processing Sympo-sium (IPDPS)
6. MonirehTaimouri* and Hamid Saadatfar. RBSEP: a reassign-ment and buffer based
7. streaming edge partitioning approach. Springer Journal of Big Da-ta (2019) 6:92
8. https://doi.org/10.1186/s40537-019-0257-5
9. Cong Xie, Wu-Jun Li, and Zhihua Zhang. S-powergraph: Streaming graph partitioning for natural graphs by vertex-cut. 2015. CoRR, abs/1511.02586.
10. Jason Riedy and David A. Bader. Massive streaming data ana-lytics: A graph-based approach. XRDS, 19(3):37. Christopher Wal-shaw. The graph partitioning archive. http://chriswalshaw.co.uk/partition/, 2016.
11. Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, 2014.
12. Wilfried Yves Hamilton Adoni , Tarik Nahhal, Moez Kri-chen2,, Abdeltif El byed and Ismail Assayad. DHPV: a distributed algorithm for large‑scale graph partitioning. Journal of Big Data (2020) 7:76. https://doi.org/10.1186/s40537-020-00357-y.
13. Charalampos E. Tsourakakis, Christos Gkantsidis, BozidarRa-dunovic, Milan Vojnovic. FENNEL: Streaming Graph Partitioning for Massive Scale Graphs. February 2014 DOI:10.1145/2556195.2556213 7th ACM international conference on Web search and data mining.
14. Makoto Onizuka, Toshimasa Fujimori, Hiroaki Shiokawa. Graph Partitioning for Distributed Graph Processing. Springer. Data Sci. Eng. (2017) 2:94–105. DOI 10.1007/s41019-017-0034-4
15. FabioPetroni, Leonardo Querzoni, KhuzaimaDaudjee, Sha-hinKamali, Giorgio Iacoboni. HDRF: Stream-Based Partitioning for Power-Law Graphs. In: Proceedings of the 24th ACM International Conference on Information and Knowledge Management (CIKM), 2015.
16. Dong Dai, Wei Zhang, Yong Chen. IOGP: An Incremental Online Graph Partitioning Algorithm for Distributed Graph Data-bases. Data Partitioning HPDC'17, June 26–30, 2017, Washington, DC, USA
17. FatemehRahimian, Amir H. Payberah, SarunasGirdzijauskas, and SeifHaridi. Distributed Vertex-Cut Partitioning. IFIP International Conference on Distributed Applications and Interoper-able Systems. DAIS 2014: Distributed Applications and Interopera-ble Systems pp 186-200
18. David Ediger Rob McColl Jason Riedy David A. Bader. STINGER: High Performance Data Structure for Streaming Graphs. 2012 IEEE Conference on High Performance Extreme Computing. DOI: 10.1109/HPEC20075.2012
19. MdAnwarulKaiumPatwary, SaurabhGarg, and Byeong Kang. Window-based Streaming Graph Partitioning Algorithm. ACSW 2019: Proceedings of the Australasian Computer Science Week MulticonferenceJanuary 2019 Article No.: 51 Pages 1–10https://doi.org/10.1145/3290688.3290711
20. Christian Mayer, Ruben Mayer, Muhammad Adnan Tariq, HeikoGeppert,
21. Larissa Laich, Lukas Rieger, and Kurt Rothermel. ADWISE: Adaptive Window-based Streaming Edge Partitioning for High-Speed Graph Processing. Date Added to IEEE Xplore: 23 July 2018. DOI: 10.1109/ICDCS.2018.00072
22. Firth, Hugo Edward Boswell. Workload-sensitive Ap-proaches to Improving Graph Data Partitioning Online. PhD Thesis. 2018. Newcastle University.
23. http://theses.ncl.ac.uk/jspui/handle/10443/4416
24. AnilPacaci and M. Tamer Özsu. 2019. Experimental Analysis of Streaming Algorithms for Graph Partitioning. In 2019 Interna-tional Conference on Management of Data (SIGMOD '19), June 30-July 5, 2019, Amsterdam, Netherlands. ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3299869.3300076
25. AydınBuluç, Henning Meyerhenke, IlyaSafro, Peter Sand-ers, and Christian Schulz.. Recent Advances in Graph Partitioning. Chapter • Research gate. November 2016. DOI: 10.1007/978-3-319-49487-6_4
26. GhizlaneEchbarthi. Big Graph Processing : Partitioning and Aggregated Querying. Databases. [cs.DB]. Université de Lyon, 2017. English. NNT : 2017LYSE1225.
27. Sahan L. Maldeniya; Ajantha S. Atukorale; Wathsala W. Vi-thanage. Network data classification using graph partition. 2013 19th IEEE International Conference on Networks (ICON). DOI: 10.1109/ICON32042.2013

28. Jiewen Huang, Daniel J. Abadi. LEOPARD: Lightweight Edge-Oriented Partitioning and Replication for Dynamic Graphs. March 2016. Proceedings of the VLDB Endowment 9(7):540-551. DOI:10.14778/2904483.2904486