Taimouri and Saadatfar *J Big Data* (2019) 6:92
https://doi.org/10.1186/s40537-019-0257-5

Journal of Big Data

RESEARCH Open Access

RBSEP: a reassignment and buffer based streaming edge partitioning approach

Monireh Taimouri* and Hamid Saadatfar

*Correspondence:
monir.taimouri@gmail.com
Computer Engineering Department, Electrical and Computer Engineering Faculty, University of Birjand, Birjand, Iran

Abstract

In recent years, the rapid growth of the Internet has led to creation of massively large graphs. Since databases have become very large nowadays, they cannot be processed by a simple machine at an acceptable time anymore; therefore, traditional graph partitioning methods, which are often based on having a complete image of the entire graph, are not applicable to large datasets. This challenge has led to the appearance of a new approach called streaming graph partitioning. In streaming graph partitioning, a stream of input data is received by a partitioner, and partitioner decides which computational machine the data should be transferred to. Often, streaming partitioner does not have any information about the whole graph, and usually distributes the vertices based on some greedy heuristics which may not be optimal for incoming vertices. Hence, partitioner's decision can be significantly improved if more information about the graph is utilized. In this paper, we present a new vertex-cut streaming graph partitioning approach. The proposed method uses the idea of postponing the decision for some of the edges (by means of an intelligent buffering) and corrects some of the past decisions to improve the quality of the graph partitioning. The proposed approach is evaluated using from real-world graphs. The experimental results show that the performance of the proposed method is superior in comparison with the previous HDRF method.

Keywords: Graph partitioning, Streaming partitioning, Edge partitioning, Big Data Processing

Introduction

In recent years rapid development of Internet has led to the emergence of large graphs. Graph partitioning is one of the useful algorithms of this new era where graphs represent connections and relationships of a real-world problem such as social networks, road networks, telecommunication networks [1] and etc. Graph partitioning is an NP-complete problem [2] and balanced edge-cut partitioning [3, 4] and vertex-cut partitioning [5, 6] are partitioning schemes that partition a graph based on the minimum edge cut and minimum vertex cut, respectively. Real-world graphs follow power-law distribution with few high degree vertices and many low degree vertices. It has been shown that edge partitioning can be more efficient for partitioning of power-law graphs [7, 8].

Emergence of large graphs introduces a new challenge where a large graph cannot be processed using non-streaming methods [9, 10] on a single machine anymore, because

the entire graph is not available and it is growing over time. To address this challenge, streaming graph partitioning [11, 12] has been proposed for large and dynamic graphs.

The HDRF (High-Degree vertices (are) Replicated First) algorithm [13] is a one-pass streaming partitioning algorithm for power-law graphs that produces balanced partitions. HDRF improves on the standard greedy approach [14] and decreases the replication factor considerably. HDRF mostly tries to balance the partitions and it may assign edges to improper partition as it cannot speculate the pattern of future edges of the graph.

In this paper, we propose the Reassignment and Buffer based Streaming Edge Partitioning (RBSEP) that not only produces balanced partitions, but also improves partitioning quality in terms of vertex-cut. RBSEP achieves this by reassigning and postponing the edge assignment through buffering mechanism where buffer space is allocated based on the available memory of a partitioner machine. Also, the decision for postponing the assignment of an edge is made based on the neighboring vertices of an edge, and if the neighborhood has not been visited yet, the edge assignment will be postponed.

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

- Proposing the idea of postponing the assignment decision for the edge which not enough neighbor vertices have been visited yet.
- Defining and employing a buffer space based on available memory of partitioner machine in order to enhance partitioning quality.
- Minor but effective correction of previous assignment decision during graph partitioning process.

Background

Problem statement

Natural graphs have a prominent property which is their skewed power-law degree distribution. It means most of the vertices have relatively few neighbors, while a few vertices have many neighbors and the probability that a vertex has degree d is

$$P(d) \propto d^{-\alpha} \tag{1}$$

where α is a positive constant that controls the skewness [14, 15]. To formally define the k-way vertex-cut partitioning problem, we represent a graph as follows: $G = (E, V)$ where V is the set of vertices and E is the set of edges, also the set of partitions is $P = (p_1, p_2, \ldots, p_k)$. Each vertex v is replicated in multiple partitions which are formed a set called A(v) ⊆ P. Finally, the goal of vertex-cut graph partitioning is to minimize the average number of vertex replicas (replication Factor) and balance size of the partitions:

$$\text{Min} \frac{1}{|\text{V}|} \sum_{\text{v} \in \text{V}} \left|\text{A(v)}\right| \quad \text{s.t.} \quad \max_{\text{p} \in \text{P}} \left|\text{E(p)}\right| < \lambda \frac{|\text{E}|}{|\text{P}|} \tag{2}$$

where λ ≥ 1 is a small constant that defines the system tolerance to load balance [7, 12]. Note that the input data is a random list of edges which are received and processed by partitioner in a streaming manner.

Related work

In the last years, partitioning of large scale graphs has become a challenging issue and has absorbed many researchers' interest. Graph partitioning is an NP-hard problem, and often solved by edge-cut and vertex-cut heuristics. On the other hand, huge growth of graphs led to appearance of streaming graph partitioning. In 2012, Stanton and Kliot [11] for the first time, proposed a streaming graph partitioning approach. In this approach a stream of graph data is received and simultaneously graph partitioning is done. One of the state of the art of streaming edge-cut partitioning approach is Fennel [16]. In this approach vertices are allocated based on the existence of the largest and smallest neighborhoods. Based on Fennel, Shi et al. [17] developed an asynchronous distributed streaming partitioning system on Apache Hadoop using map-reduce. In addition, Dong Dai et al. [18] argued it is necessary to consider the connectivity and the Vertex degree changes during graph partitioning and they designed IOGP (an incremental online graph partitioning) algorithm that responds according to the incremental changes of vertices degree. IOGP achieved better locality and generated balanced partitions while increasing the parallelism degree for accessing high-degree vertices of the graph. These researches mostly covered the problems of edge-cut-based partitioning on power-law graphs. Recently, a new heuristic greedy streaming edge-cut partitioning called HGSP [19] is introduced which creates balanced partitions that minimizing total cut edges for the constrained graph partitioning problem.

Given most of big graphs are power-law graphs and many researchers demonstrated the vertex-cut partitioning has better performance on power-law graphs [20], many heuristics have proposed in this area in the last decade. Greedy-heuristic [14] and HDRF are two of the best streaming vertex-cut partitioning algorithms where HDRF's idea of replicating highest degree vertices leads to lower replication factor value. Following HDRF, CLDA [21] takes advantage of both greedy-heuristic and HDRF algorithms. CLDA used greedy-heuristic for low-degree vertices and HDRF for other vertices. In addition, SGVCut [22], ADWISE [23] and TLP [24] are among recent edge partitioning approaches. SGVCut is a workload-aware Block-based partitioning method which tries to go beyond size balancing and to lower graph processing time by reducing the inter-partition communication. ADWISE is the window-based streaming partitioning algorithm which permits to invest more partitioning time to improve partitioning quality, and thus, reduces graph processing time. Finally, TLP is a two-stage local graph partitioning algorithm. TLP makes its partitioning decisions based on the local information instead of information from the whole graph. TLP divides the partitioning process of each partition into two stages according to modularity changes of local partitions. A different partitioning strategy is applied for each stage. HoVerCut [25] and Distributed NE [26] algorithms are also proposed with the purpose of having fast and effective partitioning. They are parallel and distributed vertex-cut partitioning which can provide higher partitioning quality in a very large scale.

Methods

As formerly mentioned, HDRF is one of the one-pass greedy algorithms which has a good performance on power-law graphs. In these kinds of algorithms, the decisions for assigning an edge will not be changed in the future. Therefore, because of lack of information about the unseen part of the graph, the decision made for placing an edge can be non-optimal. However, observing the following cases, we speculate that by postponing the assignment process of the partitioning algorithm, the performance could be improved. (1) A partitioner like HDRF assigns edges that have not been copied to any partition, to the partition with smallest number of edges only based on its balanced criterion. We believe postponing the decision for assigning an edge to the future could lead to a better partitioning. (2) Consider an edge connected to a vertex that has already been copied to a partition. A partitioner like HDRF decides to assign that edge to the partition with smallest number of edges because of its balancing purposes. This decision leads to extra copies of the corresponding vertex for future connected edges of its neighborhood. However, it is possible to increase the chance of placing the edge in its right partition by postponing its assignment process to a later time when the majority of the graph has been observed.

In this paper, we propose to extend the HDRF algorithm with a buffering mechanism that can help to postpone the edge assignment decision. Since the buffer space of the partitioner is limited, we implemented an efficient buffering strategy to determine the buffer size optimally.

Postponing the assignment decision of an edge by buffer

We consider a buffer in accordance with the partitioner's memory space. Edges enter to the partitioner and are processed by it in a streaming manner. As show in Fig. 1 the edges are categorized into four groups to have more precise buffer allocation mechanisms for them. These groups are defines as follows:

(a) Group 1: None of the vertices of the edge at both ends have been copied in a partition yet.
(b) Group 2: Just one of the end vertices of the edge has been copied in one or more partitions.
(c) Group 3: The two ends of the edge have been copied in one or more common partition(s).
(d) Group 4: The two ends of the edge have been copied in non-common partitions.

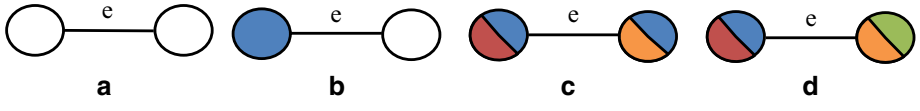

Fig. 1 Four group of the edges. Empty circles demonstrate the vertices which have not been assigned in any partitions and each color demonstrates a different partition

Strategies for inserting edges into the buffer

The buffering strategy for group 1 is to certainly transfer them to the buffer. No information about the adjacent edges of this group is available at the moment of their arrival, and thus, it is better to postpone the assignment process for them. For group 2, similar to HDRF, the new edge is allocated to one of the partitions which contain a copy of the edge's vertex; however, if the calculated score becomes maximum for the other partitions (the ones which have no copy of the edge's vertex), the edge will be transferred to the buffer. In the latter case, HDRF allocates the edge to a partition just based on the balancing factor. For the group 3 edges, there would be 3 possible cases: first, the selected partition by HDRF does not have a copy of the edges' vertices; second, the selected partition by HDRF only has a copy of one of the edge's vertices; and third, the selected partition by HDRF has a copy of the both vertices of the edge. In the first and second cases, the edge will be transferred to the buffer; while in the third case the edge would be assigned to selected partition by HDRF. Finally, the edges in group 4 would be sent to the buffer, unless the selected partition by HDRF has a copy of the either edge's

Algorithm 1 Inserting strategy into the buffer

```
Procedure Buffer-insert (e, A(v_i), A(v_j), hdrfpart)
  if A(v_i) ∩ A(v_j) ≠ ∅ then
    if hdrfpart ∈ A(v_i) ∩ A(v_j) then
      assign edge e to partition hdrfpart;
    end if
    else
      put edge e into the buffer;
    end else
  end if
  elseif A(v_i) ∪ A(v_j) ≠ ∅ then
    if hdrfpart ∈ A(v_i) ∩ A(v_j) then
      assign edge e to partition hdrfpart;
    end if
    else
      put edge e into the buffer;
    end else
  end elseif
  else
    put edge e into the buffer;
  end else
end procedure
```

vertices. Pseudocode of the edge buffering strategy is given in Algorithm 1.

The strategy for assignment an edge which is in the buffer

As mentioned in previous section, some edges may be transferred into the buffer during the graph partitioning process, because it is preferred to postpone their assignment decision to a future time when more information about the graph is available. In fact, keeping an edge in the buffer increases the chance of assigning it to a better partition which can leads to better graph partitioning totally. The partitioner will come back to process an edge inside the buffer again, after assignment of a certain amount of edges from graph stream. This way of decision making is shown in pseudo code of Algorithm 1.

Therefore, a parameter is defined for the proposed method, which is called Alpha. The value of Alpha indicates the number of edges which should be processed and assigned to the partitions before reconsidering an edge located in the buffer again. In other words, after inserting an edge into the buffer, periodically, it will be reconsidered to determine whether it could be assigned into an appropriate partition this time or it should still remain in the buffer. In addition to Alpha, another threshold is used to show that how long edges can be remained in the buffer because the buffer capacity is limited compared to a large graph. To solve this challenge, two thresholds called Yellow-line and Red-line are defined. The applications of these thresholds are showed by explaining three scenarios:

Scenario 1: Assuming that after assignment of Alpha edges, the partitioner wants to reconsider an edge inside the buffer. By checking its vertices, it will become evident that both vertices of the edge are at least in one common partition (i.e. $A(v_i) \cap A(v_j) \neq \emptyset$), afterwards, if the selected partition by HDRF is not one of these common partitions, the edge should remain in the buffer based on the pseudo code proposed in Algorithm 1. But, how long it should remain there? If the number of times that this edge is reconsidered by the partitioner reaches the Yellow-line threshold, then the edge could not remain in the buffer anymore and must be assigned to the emptiest common partition (unlike the HDRF decision). However, some reassignments are needed for this scenario which will be discussed in "The strategy for edge reassignment" section with details.

Scenario 2: Assuming the case that only one or both endpoint vertices of an edge have a copy in at least one or more non-common partitions (i.e.$A(v_i) \cup A(v_j) \neq \emptyset$). If HDRF does not decide to assign the edge to one of the partitions of $A(v_i) \cup A(v_j)$, it should be placed in the buffer. However, the Yellow-line threshold again sets a limit on the amount of times that this type of edges can remain in the buffer. It is proposed to assign the edge to the emptiest partition from the mentioned partition set ($A(v_i) \cup A(v_j)$) in these cases.

Scenario 3: assuming that none of the edge's vertices has already had a copy in any partition. In this case, the edge must remain in the buffer again unless the number of times that it is reconsidered by the partitioner reaches the Red-line threshold. If the Red-line threshold is reached, the edge is assigned to the emptiest partition in this case. The pseudo code of the edge removing strategy from buffer is given in Algorithm 2. The impact of these thresholds' value on the performance of the proposed method is evaluated in "Results and discussion" section. In general, it should be noted that the Red-line threshold must have higher value in comparison with the Yellow-line one.

The strategy for edge reassignment

This reassignment technique is used for the correction of the former decisions (but in limited cases). The purpose of using this technique is to avoid making extra copies, improving the graph partitions' integrity as well as keeping the balance among the size of the partitions. As discussed in the previous section, HDRF does not select a partition which has a copy of the edge's vertices in the first two scenarios because of their higher sizes in compared to other partitions. The idea behind the reassignment technique is to moving a few edges (if it is possible and can help to improve the partitioning quality) from the partition which has copies of vertices to the other partition. By so doing, the proposed algorithm both makes room for the new edge to be assigned to the proper partition (no extra copies created) and also correct a previous assignment which has now been found to be inadequate.

Algorithm 2 Removing strategy from the buffer

```
Procedure Buffer-remove (e,A(v_i), A(v_j), hdrfpart, counter_v_iv_j)
    if A(v_i) ∩ A(v_j) ≠ ∅ then
        if hdrfpart ∈ A(v_i) ∩ A(v_j) then
                assign edge e to partition hdrfpart;
        end if
        else
                p ←the emptiest partition in A(v_i) ∩ A(v_j)
                if counter_v_iv_j >= yellowline × Alpha then
                        Reassignment (e, p, hdrfpart);
                end if
                else
                        keep edge e in the buffer;
                end else
        end else
    end if
    else if A(v_i) ∪ A(v_j) ≠ ∅ then
        if hdrfpart ∈ A(v_i) ∪ A(v_j) then
                assign edge e to partition hdrfpart;
        end if
        else
                p ←the emptiest partition in A(v_i) ∪ A(v_j);
                if counter_v_iv_j >= yellowline × Alpha then
                        Reassignment (e, p, hdrfpart);
                end if
                else
                        keep edge e in the buffer;
                end else
        end else
    end else if
    else
         if counter_v_iv_j >= redline × Alpha then
                  assign edge e to the emptiest partition;
         end if
         else
                keep edge e in the buffer;
         end else
    end else
end procedure
```

The reassignment technique will be applied in two cases: The first case is the time that the Yellow-line threshold of an edge is reached while its two vertices have copies in some common partitions and HDRF does not select any of them. In fact, the selected partition by HDRF, in this case, has been either the one that contains only a copy of one vertex or the one that has no copy of the edge's vertices. In both cases transferring the edge to the partition which is selected by HDRF creates extra copies. The second case is the time that the Yellow-line threshold of an edge is reached while one of its vertices has a copy in some partitions which are not selected by HDRF. Assigning the edge to the partition suggested by HDRF leads to an extra copy. The reason behind these selections by HDRF despite the availability of a copy of vertices in the other partitions is the high difference between the size of the partitions that have a copy and the partition selected by HDRF.

Two different approaches are proposed for reassignment. In the first approach, it is tried to move a very small sub-graph out of the intended partition to a more appropriate one; and in the second approach, an edge that one of its endpoints vertices is placed in a partition will be moved to another partition which either has a copy of

its vertices or has a smaller size than the source partition. These two approaches are explained in more details in following sections. Algorithm 3 demonstrates the pseudo code of reassignment strategy.

Algorithm 3 Reassignment strategy

```
Procedure Reassignment (e, p, hdrfpart)
    for i ∈ number of sub-graphs of partition p
        if There is a sub-graph with size S-percent×|p| which is called S then
            f1←Subgraph-transfer (S);
            break;
        end if
    end for
    if f1≠1 then
        f2←Edge-transfer (p);
    end if
    if f2==1 or f1==1 then
        assign edge e to partition p;
    end if
    else
        assign edge e to partition hdrfpart;
    end else
end procedure
```

Algorithm 4 Reassigning a sub-graph

```
Procedure Subgraph-transfer (S)
    p ← the partition that has the maximum number of copies of sub-graph S's vertices;
    if p==-1 then
        p ← the emptiest partition;
    end if
    transfer the sub-graph S to partition p;
    return 1;
end procedure
```

Reassignment of a sub-graph

In this approach, we are looking for a sub-graph in the intended partition, which is appropriate for reassigning to another partition. This sub-graph should have a smaller size compared to the current size of the partition (for example less than 5 percent). The reason behind selecting a small size sub-graph is that such a sub-graph probably has little chance to grow while transferring it to a partition which some of its vertices have copy there not only increases the integrity in partitions but also decreases the number of the copies totally. In addition, the assignment of an edge that must leave the buffer can be done with minimum copy creation. Therefore, this reassignment decreases the replication factor. Moreover, transferring sub-graphs with larger size is not appropriate because of high computation and communication overheads.

A threshold called S-percent is defined and applied in this approach which determines that to what percentage of the partition size a sub-graph can be to be proper for reassignment. For example, if the size of a sub-graph is equal to 4 percent of the partition size (i.e. there is a sub-graph in the assumed partition which its size is equal to the

partition size × 0.04) and if the value of S-percent threshold is 5%, then the mentioned sub-graph could be suitable for transferring. Algorithm 4 demonstrates the pseudo code for reassigning a sub-graph.

Reassignment of an edge

The second reassignment approach is to transfer an edge with a degree one vertex to another partition. Therefore, when an edge must leave the buffer and be assigned to a partition which is not selected by HDRF, the proposed method searches for an edge which has a vertex with partial degree equal to one in the intended partition. The idea is to transfer this edge to another partition (which has a copy from one or both of its vertices). By this transfer a room also is made for the exiting edge from the buffer in the intended partition. This reassignment leads to a decrease in replication factor either there is a copy just of one vertex or both vertices have copies in the destination partition. Pseudo code for reassignment of an edge is given in Algorithm 5.

Algorithm 5 Reassignment of an edge

```
Procedure Edge-transfer (p)
    if There is an edge, called ed with a degree 1 vertex in partition p then
        if There is a partition, called pr that has the copy of both vertices of edge ed then
                transfer edge ed to partition pr;
                return 1;
        end if
        elseif There is a partition, called pr that has the copy of one of vertex of edge ed then
                transfer edge ed to partition pr;
                return 1;
        end elseif
        else
                return 0;
        end else
    end if
    else
          return 0;
    end else
end procedure
```

Results and discussion

The purpose of this section is to evaluate the performance of RBSEP in comparison with HDRF and two other algorithms, greedy-heuristic and CLDA. For fulfilling this purpose, the proposed algorithm, which consists of the partitioner, buffering and reassignment functions are implemented and their performances on the real and large graphs, are evaluated. The proposed algorithm is implemented in the Visual Studio by C++. Moreover, for having a fair evaluation, the other algorithms are also implemented in C++.

The graphs used in the experiments are taken from SNAP [27] and KONECT [28] Websites. All the graphs are undirected and their edges are without any weights. These graphs are usually following power-low distribution and are contextual files. Each line of these files includes the information of the edges. The real-world graphs are shown in Table 1.

Table 1 The real-world graphs

Dataset	$\|V\|$	$\|E\|$
Hamsterster	1.858 K	12.534 K
Ego-Facebook	4.039 K	88.234 K
Astro-Ph	18.771 K	198.05 K
Douban	154.908 K	327.162 K
WordNet	146.005 K	656.999 K
com-DBLP	317.08 K	1.049866 M

Time and space complexity analysis

The proposed approach, as mentioned before, processed most edge of the graph only once. However, a few edges whose neighborhoods have not yet been visited will be moved to the buffer. In the worst case, these edges should be processed as many times as the Red-line threshold indicates. Experiments show that this delay in assignment of these edges increases the quality of partitioning. But in very few cases, this is not enough to delay decision and some of the earlier decisions should be also amended. In fact, this situation occurs when the partition with the largest number of neighbors on an edge is crowded and the proposed approach tries to correct a past decision by moving one resident edge of this partition to a more proper one. For doing this, the partition which has higher number of copies of edge's vertices should be searched. Based on the details described about the proposed algorithm its computational complexity is $O(n+m*r)+O(p*l*k\log k)$, where n is the number of edges, m is the number of edges that entered into the buffer and p is the number of reassignments. Likewise r is Red-line, k is the number of partitions and l is the partition size.

The space complexity of the proposed approach is as much the same as the HDRF method. The partitioner needs to maintain the partial degree of the visited vertices just like the HDRF algorithm. However, a fixed size buffer space based on the available memory of the partitioner machine is also allocated and used by the proposed algorithm.

Criteria for evaluating the partitioning quality

For showing the partitioning quality of a vertex-cut approach, two parameters are usually used: replication factor and balance factor. Replication factor is a criterion for evaluating the expenses of communication among distributed systems and indicates the average number of copies of vertices in the system. The formula for calculation of replication factor is as follows:

$$RF = \frac{1}{|V|}\sum_{v\in V}|A(v)| \tag{3}$$

where V is the total number of the vertices and $A(v)$ is a set includes all the partitions in which the vertex v is copied. The closer to 1 the replication factor value is, the higher the partitioning quality would be.

Balance factor is another criterion for analyzing the graph partitioning process. The following formula is used for the calculation of balance factor:

$$\max_{p\in P} |E(p)| < \lambda \frac{|E|}{|P|} \tag{4}$$

where E is the total number of edges, P is the total number of partitions, $E(p)$ is the number of edges assigned to p and λ is a parameter which is determined by the user and shows the degree of acceptable deviation from the balance. The value of λ should be real ($\lambda \in R$) and greater than 1 ($\lambda > 1$). The above inequality shows the acceptable value for $E(p)$ in comparison to the number of expected edges of each partition ($\frac{|E|}{|P|}$). Thus, the following equation can be written for λ parameter:

$$\lambda = \frac{|P|}{|E|} \max_{p\in P} |E(p)| \tag{5}$$

where λ is the balance factor and used as a criteria for evaluation of the graph partitioning quality.

Evaluation of the proposed method

In the following sections, the performance of the proposed method is discussed and evaluated compared with HDRF, greedy-heuristic and CLDA based on the replication and balance factors. In addition, the impact of the value of the RBSEP's defined parameters (such as Red-line) on the graph partition quality was explored.

The comparison of replication factor of RBSEP and HDRF

In these experiments, the values of the thresholds have been fixed. The value of Alpha is equal to 5, Yellow-line is 1, Red-line is 4 and the sub-graph parameter is assumed to be 0.1.

In Fig. 2, the comparison of the replication factor of the proposed method and three mentioned algorithms for the mentioned graphs is shown. As it can be seen, by increasing the number of the partitions, applying the proposed graph partitioner led to lower replication factor values in comparison to the other algorithms. As the number of the partitions increases, the proposed method has wider functional range and can better show its capability. Despite the fact that the 'Douban' graph does not follow 'Power-law' distribution, but the functional superiority of the RBSEP over the other algorithms is indicated in the Fig. 2d.

The impact of balancing on the performance of the algorithms

One of the effective factors on the behavior of a graph partitioning algorithm is the degree of balancing. In general, a graph can be partitioned with lower vertex copies when less balance between the partitions must be met. In order to investigate the impact of the balance level on the functionality of the proposed algorithm, its performance for different ranges of balance factor was evaluated and compared to HDRF and two other algorithms. For the experiments of this section, 'Facebook' graph has been used.

By increasing the balance level in HDRF, greedy-heuristic and CLDA, the replication factor increases; while for RBSEP, the replication factor remains almost the same (Fig. 3a). The reason behind this result is the reassignment ability of the proposed method.

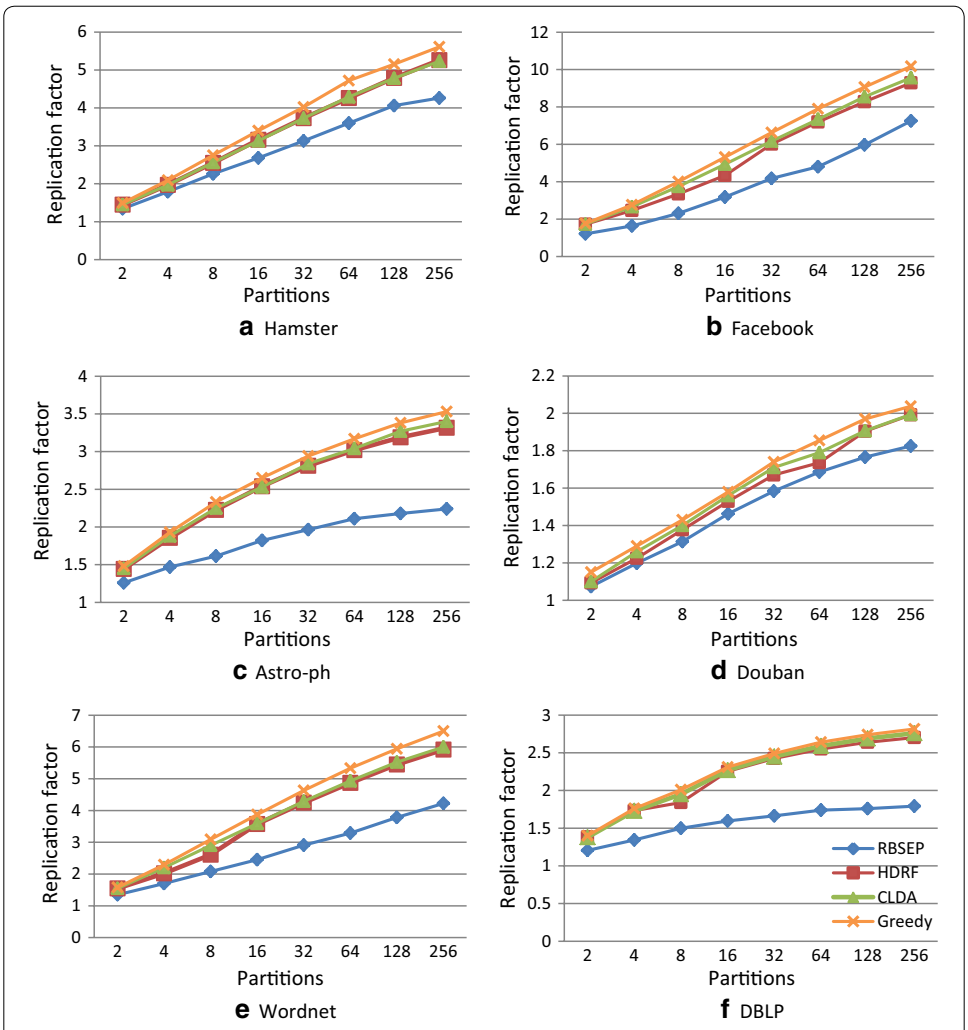

Fig. 2 Comparison of replication factor in PBSEP and the other algorithm for variant graphs

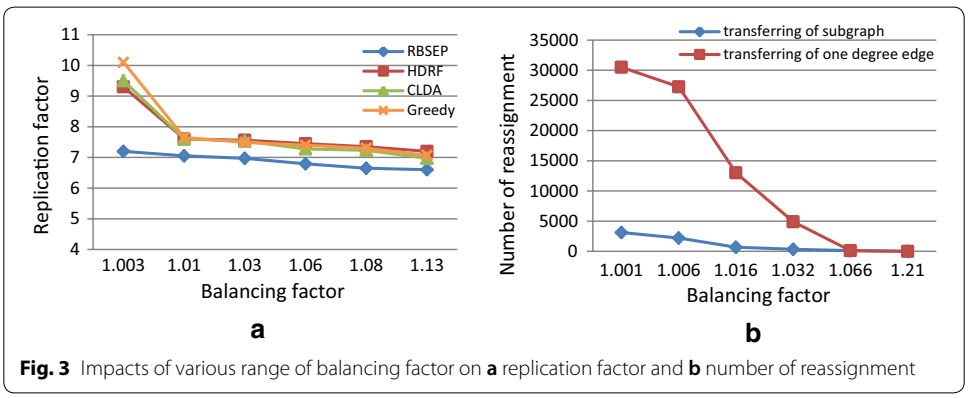

Fig. 3 Impacts of various range of balancing factor on **a** replication factor and **b** number of reassignment

Considering less limitation in keeping balancing, more candidate partitions can be considered for edges to be placed in; and thus, there is less necessity for reassignment so that the number of reassignment reaches zero for balancing factor equal to 1.066. As

shown in the Fig. 3b, by increasing the replication factor, the need for reassignment is extremely diminished.

The impact of the thresholds on replication factor and reassigning

As mentioned earlier, for controlling the behavior of the proposed algorithm, some parameters are defined. These parameters determine some thresholds in buffering and reassignment processes. In this section, the impact of these thresholds on the functionality of the proposed algorithm is discussed. For the experiments presented below, 'Hamsterster' graph has been used.

The impact of the Red-line threshold on replication factor and reassigning As it was mentioned in "Methods" section, the edge that has no copied vertices yet in any partitions must wait in the buffer unless the time it spent in buffer exclude the Red-line. In such situation, it must be assigned to the emptiest partition. Figure 4a indicates that by increasing the value of the Red-line parameter, replication factor decreases. The reason for this issue is that higher values of the Red-line threshold give more chance to an edge for remaining in the buffer and consequently higher opportunity for placing in the partition where at least one of the edge's vertices has been copied before. However, it is apparent that by increasing the value of this parameter, the volume of computations increases as well. Therefore, the suitable value for this parameter depends on the application requirements. However, since the graph is partitioned once and the result of this partitioning is used several times, thus the quality of partitioning is preferred to a bit more computation generally.

Figure 4b shows the number of reassignment for different values of Red-line threshold. As it is shown, by increasing the value of the Red-line parameter the number of reassignment increases as well. Therefore, higher value of Red-line means lower replication factor (based on Fig. 4a) and higher number of reassignment. In fact, there is trade off here between the number of the copies and the computation overhead. Consequently, the Red-line parameter has a controlling role in the proposed method and can be tuned properly based on the partitioning goals.

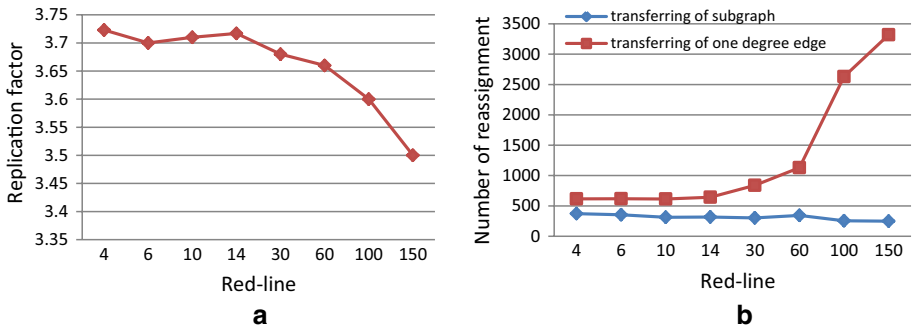

Fig. 4 Impacts of various range of Red-line on **a** replication factor and **b** number of reassignment

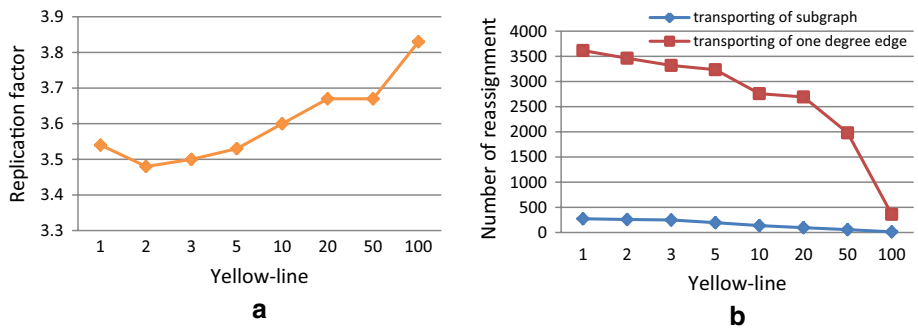

Fig. 5 Impacts of various range of Yellow-line on **a** replication factor and **b** number of reassignment

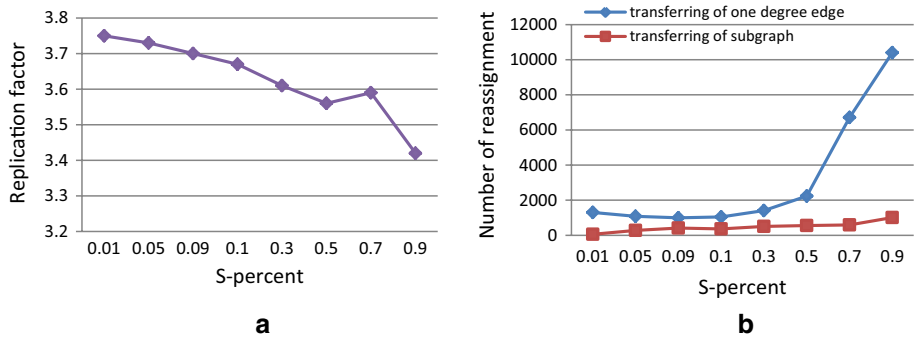

Fig. 6 Impacts of various range of Yellow-line on **a** replication factor and **b** number of reassignment

The impact of the Yellow-line threshold on replication factor and reassigning This threshold determines that how many Alpha times the edges (which at least one of their vertices has a copy in a partition), could remain in the buffer. This group of the edges is the ones that should be assigned to the partition which at least has a copy of one of their vertices, but HDRF decides to assigns them to another partition because of balancing; however, RBSEP postpone the decision for these edges by putting them into the buffer. Figure 5a shows the effect of Yellow-line threshold value on the replication factor criteria. As it can be seen, the replication factor generally increases by growth of the Yellow-line threshold value. Nevertheless, the total changes are not considerable.

Figure 5b shows the number of the reassignment for different values of the Yellow-line threshold. The increment of this threshold leads to a decrease in the number of reassignment. However, based on the results presented in Fig. 5a, the replication factor does not change significantly. Therefore, it could be noted that although postponing the assignment of this kind of edges do not affect the replication factor noticeably; the number of reassignment is decreased greatly.

The impact of the sub-graph size on replication factor and reassigning By increasing the size of the sub-graph parameter, the number of reassignment also increases and the replication factor decreases in contrary. Figure 6a shows the reduction of the replication factor by increment in the sub-graph size parameter.

As discussed above, it is expected that increment in the value of sub-graph size parameter leads to less number of reassignment. The sub-graph size parameter called 'S-percent' is defined as a percent of the partitions' size. Experiment indicated that higher value of 'S-percent' causes an increase in the number of reassignments (see Fig. 6b). This threshold is also a tool in the hands of the partitioner for balancing between computation expenses and replication factor. However, the value of 0.1 seems to be reasonable for the 'S-percent' parameter (suggested based on the results).

Performance of RBSEP for different input volumes

As it can be seen in Fig. 7, the proposed approach can better show its ability in graph partitioning for graphs with larger size. The proposed approach is able to reach lower replication factors, in general, for larger input volumes. Furthermore the number of partitions will not have significant negative effect on this ability of the proposed approach especially in the case of larger graphs. In fact, when the size of the input graph becomes larger, the chance that the vicinity of an incoming edge has been visited already decreases.

As a result, the importance of the idea of postponing edge assignment decisions can be more prominent. In addition, making the wrong decisions is more common in large graphs. For this reason, the ability to correct past wrong decisions has made the proposed method suitable for large scale graphs.

Conclusion

Since the streaming partitioning decisions are made based only on the observed part of the graph, they might be non-optimal. Therefore, we have tried to improve the partitioning quality by postponing the decision for edges which cannot be allocated properly because of lack of information and reassigning some of the edges formerly allocated incorrectly.

Therefore, we considered two ideas for achieving these goals. First, a buffer is considered according to the partitioner's memory space. When an edge enters the partitioner and there is no information about neighboring edges, it will be kept in the buffer until more information becomes available and a better assignment can be achieved.

In addition, the proposed reassignment technique helps improve the partitioning quality. The idea is to reassign an edge which previously allocated to a partition incorrectly

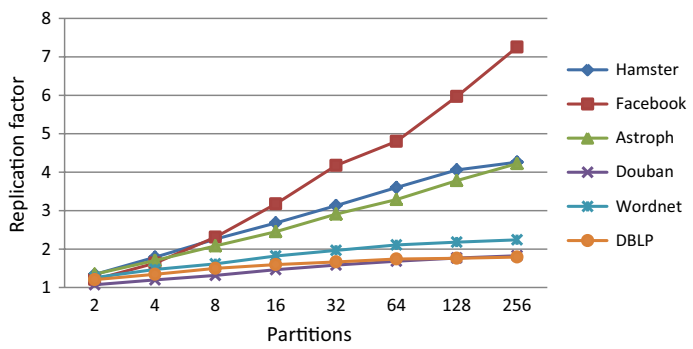

Fig. 7 Performance of RBSEP on large scale graph

because of balancing or incomplete information, to a more appropriate one. As experimental results show, the proposed method has decreased the replication factor further than HDRF while keeping the partitions' size balanced.

Abbreviations
RBSEP: Reassignment and Buffer based Streaming Edge Partitioning; HDRF: high degree (are) replicated first; IOGP: incremental online graph partitioning; HoVerCut: horizontally and vertically vertex-cut partitioner; ADWISE: Adaptive Window-based Streaming Edge Partitioning; RF: replication factor; CLDA: consider low-degree edge assignment; Distributed NE: distributed neighbor expansion; TLP: two-stage local partitioning.

Authors' contributions
MT and HS contributed to the process of reviewing and analyzing the literatures of the research, to the design, implementation and evaluation of the proposed approach and finally the writing of the manuscript. Both authors read and approved the final manuscript.

Authors' information
Monireh Taimouri has received her B.Sc. degree in Information Technology Engineering at University of Sistan and Baluchestan in 2012 and her M.Sc. degree at University of Birjand in 2018. Her research interests include parallel and distributed computing, big data analysis and graph theory.

Hamid Saadatfar is currently an assistant professor of Computer Department at University of Birjand. He has received his B.Sc., M.Sc., and Ph.D. degrees from Ferdowsi university of Mashhad in 2007, 2009 and 2013, respectively. His research interests include cluster and grid computing, distributed data mining, big data analysis and power aware computing.

Funding
The authors declare that they have not used any source of funding for this research.

Availability of data and materials
All datasets used in our research are available via following URL addresses:
http://snap.stanford.edu/data, http://konect.uni-koblenz.de/networks/.

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Received: 27 July 2019 Accepted: 5 October 2019
Published online: 19 October 2019

References

1. Margan D, Pietzuch P. Large-scale stream graph processing: doctoral symposium. In: Proceedings of the 11th ACM international conference on distributed and event-based systems 2017, p. 378–381. ACM.
2. Andreev K, Racke H. Balanced graph partitioning. Theory Comput Syst. 2006;39(6):929–39.
3. Huang J, Abadi DJ. Leopard: lightweight edge-oriented partitioning and replication for dynamic graphs. Proc VLDB Endow. 2016;9(7):540–51.
4. Hao Y, Li G, Yuan P, Jin H, Ding X. An association-oriented partitioning approach for streaming graph query. Sci Program. 2017;2017:1–11. https://doi.org/10.1155/2017/2573592.
5. Cao, Y., Rao, R.: A streaming graph partitioning approach on imbalance cluster. In: Advanced Communication Technology (ICACT), 2016 18th International Conference on 2016, p. 360–364. IEEE.
6. Xie C, Li W-J, Zhang Z. S-powergraph: STREAMING graph partitioning for natural graphs by vertex-cut. arXiv preprint arXiv:1511.02586 (2015).
7. 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 2014, p. 1673–1681.
8. Chen R, Shi J, Chen Y, Chen H. PowerLyra. Paper presented at the proceedings of the tenth european conference on computer systems-EuroSys '15.
9. Karypis G, Kumar V. A parallel algorithm for multilevel graph partitioning and sparse matrix ordering. J Parallel Distrib Comput. 1998;48(1):71–95.
10. Rahimian F, Payberah AH, Girdzijauskas S, Haridi S. Distributed vertex-cut partitioning. In: IFIP International conference on distributed applications and interoperable systems. New York: Springer; 2014. p. 186–200.
11. 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. ACM. p. 1222–30.
12. Abbas Z, Kalavri V, Carbone P, Vlassov VJ. Streaming graph partitioning: an experimental study. Proc VLDB Endow. 2018;11(11):1590–603.

13. 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. ACM; 2015, p. 243–252.
14. Gonzalez JE, Low Y, Gu H, Bickson D, Guestrin C. Powergraph: distributed graph-parallel computation on natural graphs. In: OSDI 2012, vol. 1, p. 2.
15. Onizuka M, Fujimori T, Shiokawa H. Graph partitioning for distributed graph processing. Data Sci Eng. 2017;2(1):94–105.
16. 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. ACM; 2014, p. 333–342.
17. Shi Z, Li J, Guo P, Li S, Feng D, Su Y. Partitioning dynamic graph asynchronously with distributed FENNEL. Future Gener Comput Syst. 2017;71:32–42.
18. Dai D, Zhang W, Chen Y. POSTER: IOGP: an incremental online graph partitioning for large-scale distributed graph databases. In: ACM SIGPLAN Notices 2017, vol. 8, ACM; 2017. p. 439–40
19. Li Q, Zhong J, Cao Z, Li X. Optimizing streaming graph partitioning via a heuristic greedy method and caching strategy. Optim Methods Softw, 1–16 (2019).
20. Verma S, Leslie LM, Shin Y, Gupta I. An experimental comparison of partitioning strategies in distributed graph processing. Proc VLDB Endow. 2017;10(5):493–504.
21. Rad HC, Azmi R. CLDA: vertex-cut partitioning for streaming power-law graphs. In: 2017 9th international conference on information and knowledge technology (IKT). IEEE; 2017, p. 104–10.
22. Li Y, Constantin C, Mouza C. Sgvcut: a vertex-cut partitioning tool for random walks-based computations over social network graphs. In: Proceedings of the 29th international conference on scientific and statistical database management. ACM; 2017, p. 39.
23. Mayer C, Mayer R, Tariq MA, Geppert H, Laich L, Rieger L, Rothermel K. ADWISE: adaptive window-based streaming edge partitioning for high-speed graph processing. In: 2018 IEEE 38th international conference on distributed computing systems (ICDCS). IEEE; 2018, p. 685–95.
24. Ji S, Bu C, Li L, Wu X. Local graph edge partitioning with a two-stage heuristic method. In. EasyChair; 2019.
25. Sajjad HP, Payberah AH, Rahimian F, Vlassov V, Haridi S. Boosting vertex-cut partitioning for streaming graphs. In: Big Data (BigData Congress), 2016 IEEE international congress on 2016. IEEE. p. 1–8.
26. Hanai M, Suzumura T, Tan WJ, Liu E, Theodoropoulos G, Cai W. Distributed edge partitioning for trillion-edge graphs. arXiv preprint arXiv:1908.05855 (2019).
27. Leskovec J, Krevl A. Stanford large network dataset collection. http://snap.stanford.edu/data (2014). 2018.
28. Kunegis J. KONECT—The Koblenz Network Collection. http://konect.uni-koblenz.de/networks/ (2013). 2018.

Publisher's Note