SWR: Using Windowed Reordering to Achieve Fast and Balanced Heuristic for Streaming Vertex-Cut Graph Partitioning

Jie Wang and Dagang Li(✉)

School of ECE, Shenzhen Graduate School, Peking University, Shenzhen, China
wang_jie@pku.edu.cn, dagang.li@ieee.org

Abstract. Graph partitioning plays a very fundamental and important role in a distributed graph computing (DGC) framework, because it determines the communication cost and workload balance among computing nodes. Existing solutions are mainly heuristic-based but unfortunately cannot achieve partitioning quality, load balance, and speed at the same time. In this paper, we propose Sliding-Window Reordering (`SWR`), a streaming vertex-cut graph partitioning algorithm, that introduces a pre-partitioning window to re-order incoming edges, making it much easier for a greedy strategy to maintain balance while optimizing edge assignment at a minimal computational cost. We analytically and experimentally evaluate `SWR` on several real-world and synthetic graphs and show that it achieves the best overall performance. Compared with `HDRF`, the state-of-the-art at present, the partitioning speed is increased by 3–20 times, and the partitioning quality is increased by 15% to 30% on average when achieving balanced load among all nodes.

Keywords: Graph partitioning · Streaming algorithms · Vertex-cut · Distributed graph-computing frameworks · Load balancing

1 Introduction

In recent years we have seen a fast growth in large-scale graph-structured data in various real-world applications. In order to process these large scale datasets, distributed graph-computing (DGC) frameworks become favorable which partition the graph across multiple machines that can compute in parallel using graph processing systems such as Pergel [16] or GraphLab [15]. Different graph partitioning (GP) strategies can greatly affect the overall performance of the

This work was supported by Shenzhen Peacock Innovation Program (KQJSCX20180323174744219), National Engineering Laboratory for Video Technology - Shenzhen Division and the Shenzhen Municipal Development and Reform Commission (Disciplinary Development Program for Data Science and Intelligent Computing).

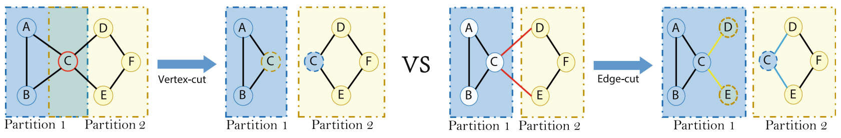


Fig. 1. Vertex-cut vs. edge-cut graph partitioning

DGC framework, because they determine the communication cost and workload balance among these machines.

Graph partitioning strategies can be categorized into either vertex-cut or edge-cut approaches. They both replicate some vertices into different partitions. Figure 1 shows how they work with a simple example. The number of edges in each partition determines the workload of each machine, and the total number of replicas determines the total communication cost to synchronize them.

Graph partitioning can be generally regarded as typical balanced k-way partitioning problem, while both vertex-cut and edge-cut variants are known to be NP-hard [2,7,13]. Although many algorithms have been proposed to solve the problem, most of them have high cost and cannot process large-scale real world graphs. Streaming graph partitioning has been proposed by Stanton and Kliot [19] to partition large scale graphs fast and efficiently. Since the quality of GP is crucial to the overall performance of DGC frameworks, adequate GP algorithms should be developed, taking three important aspects into consideration. The first is quality, mostly referring to the communication cost for synchronizing replicated items across different partitions during the computation phase. The second is balance. Large workload skews in a cluster of machines can greatly deteriorate the efficiency of the system. The third is speed. Although some existing GP algorithms can achieve quality partitions, if the GP phase itself takes too much time, the overall performance is still affected, for example off-line algorithms such as JA-BE-JA-VC [18], METIS [12], Ginger [3] and NE [21].

In this paper we focus on the edge distribution of the graph dataset since real world graphs are known to have skewed power-law degree distribution [8], and the order of incoming edges in the stream also show some common patterns. Several studies [1,14] have shown that edge-cut methods do not work well with power-law graphs, while vertex-cut is proven to be the better choice in theory [6] and in practice [8,9]. We will first analyze in depth two representative vertex-cut GP algorithms, `greedy` and `HDRF`, and then propose our *sliding-window reordering* (`SWR`) mechanism that introduces sliding-window based pre-reorder to the input edges as well as enhanced greedy rules, so the degree and neighboring information are better utilized to achieve better partitioning performance in terms of speed, quality and balance. The contributions are as follows:

1. A clustering model is provided to describe the characteristics of the input edges, and in-depth analysis on standard `greedy` and `HDRF` is provided to show how they are affected by the order of the edges.

2. A new streaming vertex-cut GP algorithm called `SWR` is proposed, which achieves better partition quality, speed and balance at the same time.
3. A new concept of dispersion factor is proposed and used to analyze theoretically the size of the window on the effectiveness of the SWR algorithm.

2 Problem Definition and Background

2.1 Balanced *k*-Way Vertex-Cut GP Problem

Consider a graph $G = (V, E)$ consisted of vertices $V = (v_1, v_2, \cdots, v_n)$ and edges $E \subseteq V \times V$. Vertex-cut GP will assign the edges into a series of pairwise disjoint sets $P = (p_1, p_2, \cdots, p_n)$, where for any $i \neq j$, $p_i, p_j \subseteq E$ and $p_i \cap p_j = \emptyset$.

Since in vertex-cut, cut vertices will have replicas in multiple partitions, we define $A(v)$ as the set of partitions where vertex v is replicated. During computation these replicas need to be synchronized which generates communication cost, therefore the main target of vertex-cut is to minimize replications while keeping partitions of each machine approximately the same size, so the balanced $|P|$-way vertex-cut GP problem can be formally defined as to solve:

$$minimize \frac{1}{|V|} \sum_{v \in V} |A(v)| \quad s.t. \quad \max_{p \in P} |p| < \delta \frac{|E|}{|P|} \tag{1}$$

where $\delta \geq 1$ is a small constant that defines the tolerance to partition imbalance. The object function in Eq. 1 is called **Vertex Replication Factor (VRF)** representing the average number of replicas per vertex.

2.2 Rule-Based and Score-Based Approaches

The core of a GP algorithm is the strategy to assign which edge to which partition. Most existing GP algorithms can be generally divided into two categories: rule-based and score-based. In a rule-based strategy, new edge is assigned to the partition that satisfies a number of simple rules, so the whole assignment process can be carried out with a series of if-else questions. For score-based strategy, matching scores need to be calculated between the new edge and all existing partitions, and the one with the highest score will be chosen, therefore much more calculation is involved in assigning each single edge. Rule-based approaches are generally faster than score-based ones but less optimal in partitioning quality. However they are not completely opposite to each other: rule-based strategy can be transformed to score-based, one example is the formal transformation of `greedy` in [17], but score-based algorithms are not always transformable to rule-based because score computation can be rather complicated.

2.3 Streaming Vertex-Cut Partitioning Algorithms

In streaming partitioning, a GP algorithm will process over the edge stream, which means that it needs to partition all the edges in a single pass. Streaming partitioning is fast and easy to be integrated to DGC frameworks because of its simplicity. However, good partitioning quality is more difficult to achieve because partitioning decisions are made on limited information, and minimizing vertex replication can easily end up in severe partition imbalance. If you want both quality and balance it will be too slow. To the best knowledge of the authors, there is not yet a solution that performs well in all the three important aspects.

Hashing Partitioning Algorithms: Hashing GP algorithms are rule-based. It can achieve fast and balanced partitioning but has no quality guarantee. The most known hashing GP are `hashing` in GraphX and `DBH` [20]. Because no partitioning history is considered in `hashing`, it results in very high VRF, especially on power-law graphs. `DBH` improves over pure hashing by taking vertex degree into consideration and cut higher degree vertices first, so the VRF is reduced.

Constrained Partitioning Algorithms: This category of GP algorithms are also rule-based. The core idea is to limit the candidate partitions a vertex can go within a *constrained set*, so as to restrain the VRF with a theoretical upper bound. The more constrained the rules are the better the quality could be but at higher risk of severe imbalance. Typical constrained GP algorithms are `grid` and `PDS` from GraphBuilder [11]. In `grid`, the partitions are represented in a matrix $P = M \times N$. Hash function $h(v)$ maps vertex v onto this matrix, and the constrained set $S(v)$ is the subset of partitions in P that are at the same row and column as v. The candidate partitions for the edge would be the two intersect partitions of $S(v_i)$ and $S(v_j)$ from its two vertices. `PDS` generates constrained sets using Perfect Difference Sets [10] which is more constrained.

Greedy Partitioning Algorithms: Greedy algorithms maintain some state information of the partitioning history to help later assignment. Simply put, vertex v is preferred to be assigned to a partition that it has been assigned before with earlier edges, and the smallest one is chosen when multiple partitions are qualified to improve balance. The `greedy` algorithm from PowerGraph [8] is rule-based and also the basis for many other more advanced greedy GP algorithms. `HDRF` is the state-of-the-art greedy algorithm which is score-based and can not be transformed to fast rule-based form. It follows the same greedy strategy but also take the vertex degree into consideration by calculating a matching score between the incoming edge and all existing partitions. A balance parameter λ is used to control the imbalance degree in the score calculation. `ADWISE` goes one step further by evaluating scores of *a window of* edges in the stream at the same time and assign the edge with the best score to the corresponding partition. Although score-based greedy algorithms may get improved partitioning quality, they are much slower, especially when existing partitions are big and plenty.

3 Edge Stream Model and Its Impacts on GP Algorithms

Natural graphs from the real-world are commonly found to be highly skewed with power-law degree distribution. Furthermore, the edge order very often resembles some specific patterns when loaded as edge stream, for example clustered in groups by the source vertex, or in the order of a BFS, etc., all tracing back to how the graph data were collected or stored in the first place. We found that these patterns in the edge order have strong impact on the partitioning results.

3.1 Clustering Model of Edge Stream

First we propose a clustering model on the ordering pattern of an edge stream to discuss its impacts on GP algorithms. A *dispersion factor* is defined to measure how strong consecutive edges are related to each other to resemble a pattern. The larger the dispersion factor the less clustering effect the edges have and the weaker the order pattern.

As a building block to the clustering model we define a **self-contained edge stream**, which can be represented by an ordered set of edges $E_{std} = \{e_1, e_2, \cdots, e_n\}$, where for any $j > 1$ exists $i < j$ satisfying $e_i \cap e_j \neq \emptyset$, meaning a later edge e_j always shares a common vertex with some earlier edge e_i. A self-constrained edge stream can be for example a sequence of edges collected by a single-thread breadth-first search from a connected graph. In this specific example of BFS, we further have: for any $e_j = (u_j, v_j), j > 1$ there exists $e_i = (u_i, v_i), i < j \textit{ and } i \geq 1$, such that $v_i = u_j$. A self-contained edge stream shows very strong clustering effect since all the edges are related to some of the earlier edges in the stream, as if they are "dragged in" by their neighbors.

Considering that real-world graphs are most collected by multiple crawlers, combined from various sources or read from a distributed storage, although edges are more mixed, we still observe apparent clustering effect. To describe these more-general graphs, we can regard their edge stream as constructed from m intertwined self-constrained edge sub-streams. Essentially, edge stream from any graph dataset can be regarded this way, the only difference is how big m should be to describe it. Therefore, we choose m to define the **dispersion factor (DF)** of an edge stream. Formally put, any edge stream E can be defined as:

$$E = E_1 \cup E_2 \cup \cdots \cup E_m \tag{2}$$

where $E_i, i \in \{1, \cdots, m\}$ are self-contained edge streams.

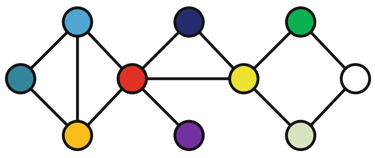
Furthermore, if E and E_i, E_j are defined as

$$E_i = \{e_{i_1}, e_{i_2}, \cdots, e_{i_m}\}, E_j = \{e_{j_1}, e_{j_2}, \cdots, e_{j_k}\} \subset E = \{e_1, e_2, \cdots, e_n\} \tag{3}$$

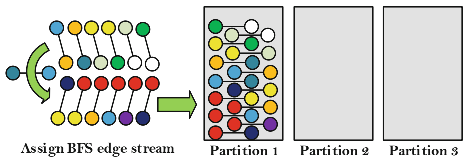
then when $e_{i_m} \in E_i = e_t \in E$, there exists $i_k < i_m, r < t$ satisfying $e_{i_k} \in E_i = e_r \in E$, which means that the edges of E_i maintain their relative order in E. For edges from two different self-contained sub-streams E_i and E_j, we may have $e_i \in E_i \cap e_j \in E_j = \emptyset$ or not, since E_i and E_j may come from disconnected subgraphs, or different parts of a connected subgraph but only get to each other later on in the stream. They can not be merged according to the definition.

3.2 Impacts on Rule-Based Greedy

The rule-based `greedy` is fast and provides rather good partitioning quality, but for datasets with intrinsic small DF it may have very severe imbalance problem.



(a) A sample connected graph



(b) `greedy` partitions all edges of a BFS stream into a single partition

Fig. 2. A partitioning example with `greedy`. (Color figure online)

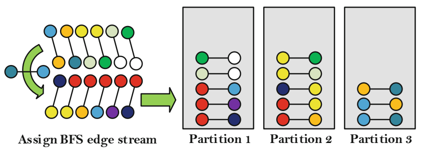
Let P_k be the k-th partition, $V(P_k)$ be its vertex set and $E(P_k)$ be its edge set. $e_i = (u_i, v_i)$ is the i-th edge of the edge stream.

We first analyze `greedy` on a self-contained edge stream E_{std} with DF of 1. For the first edge $e_1 \in E_{std}$, we can assign it to any of the empty partitions, let us say P_1, so we have $e_1 \rightarrow P_1$. For any later edge $e_j \in E_{std}$ *and* $j > 1$, according to the `greedy` rules, we need to see if any partition P_k already has u_j *or* $v_j \in P_k, k \in \{1, \cdots, m\}$. For the self-contained edge stream, there must exist $i < j$ satisfying $e_i \cap e_j \neq \emptyset$. Since the earlier edge e_i has already been assigned to a partition P_i, e_j will be assigned to that same partition, so we have $e_j \Rightarrow e_i \rightarrow P_i$. For the same reason, when e_i was assigned, there must have existed $h < i$, so that $e_i \Rightarrow e_h \rightarrow P_h$. In the end we can only have $e_j \Rightarrow e_i \Rightarrow e_h \Rightarrow \cdots \Rightarrow e_1 \rightarrow P_1$, and all edges of E_{std} are assigned to the same partition P_1. A demonstrative example is shown in Fig. 2(b), where the self-contained edge stream comes from a BFS starting from the red vertex in Fig. 2(a). The situation is less severe but similar for an edge stream with a larger DF of m. Since $E = E_1 \cup E_2 \cup \cdots \cup E_m$, if for any $i \neq j$, all $e_{i_r} \in E_i$ and $e_{j_t} \in E_j$ satisfy $e_{i_r} \cap e_{j_t} = \emptyset$, then every $E_k, k \in \{1, \cdots, m\}$ will have all its edges assigned to a separate partition of its own in the way described above, so the graph will be naturally partitioned into m partitions. If m is smaller than the target number of partitions n, we will end up with $(n - m)$ empty partitions; even when m is larger than n but not much, keeping the n partitions in balance is still difficult if possible.

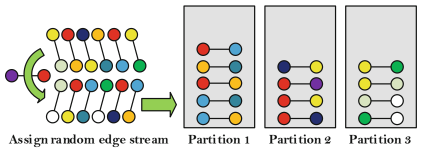
Of course in reality, it is likely that some E_k's will have common vertices later on down the stream. Such situation will not change the DF m of the edge stream, but it may help to get more evenly distributed partitions, for some edges now have more choices during the assignment, `greedy` can use that to fill up smaller partitions when appropriate. That is why in practice `greedy` can still achieve acceptable balance on most natural graphs with moderate clustering effect.

3.3 Impacts on Score-Based Greedy

`HDRF` is the state-of-the-art GP algorithm that can achieve both low VRF and good balance. However, it handles the imbalance problem by introducing a balance parameter λ which enforces a brutal regulation to penalize edge assignment choices that may exceed the imbalance tolerance. Such methodology works but also brings new problems. First, a self-contained sub-stream may be brutally split



(a) HDRF has sub-optimal edge assignment.



(b) Re-ordering edges results in better partitioning.

Fig. 3. A partitioning example with `HDRF` and Re-ordering.

into multiple partitions so that its edges are sub-optimally assigned because of the balance enforcement. Second, the introduction of λ means that `HDRF` can only be score-based, and the involved score computation with all existing partitions to assign each new edge affects its speed a lot.

A simple example can demonstrate why sub-optimal edge assignment may happen. The same BFS edge stream from Fig. 2(a) is reused here, whose imbalance tolerance is set to 2. From Fig. 3(a) we can see much more balanced partitions, which is to be expected; however, the price paid is an increase in vertex replication, because now some edges with common vertices have to be assigned to different partitions to satisfy the balance requirement. As an indication, VRF also goes to 1.6 from 1 as in Fig. 2(b).

4 Design of `SWR` Algorithm

In this section we will introduce our `SWR` algorithm, which is designed for real-world big graph datasets that generally always possess a certain level of clustering effect. The discussion will be carried out from the aspects of load balance, quality and speed, respectively, and the pseudo code is shown in Algorithm 1.

The main idea is to introduce a moving-window mechanism on the edge stream, and by changing the order of the edges inside the window, we can attenuate the clustering effect and increase the DF, which makes a rule-based greedy heuristic to easily achieve partitioning balance while assigning edges optimally, taking the vertex degree and neighboring information into consideration.

4.1 Using Randomization to Keep Balance

Randomization can help to keep balance. To demonstrate the idea we refer back to the same graph in Fig. 2(a) but now pick the edges one by one at random to form a new edge stream, and use the same `greedy` algorithm to process them. The result is a set of balanced partitions as shown in Fig. 3(b) with a VRF of 1.2. Intuitively, a randomized edge stream has a smaller clustering effect and larger DF. We will analyze it theoretically.

To see that let us start with a completely random edge stream containing $|V|$ vertices and $|E|$ edges, and the average probability of vertex appearing in a new edge is $\frac{2}{|V|}$. During a streaming GP process, if none of the vertices of a new edge has been seen before, a new self-contained sub-stream will be created. When m edges have been processed, the probability that a new self-contained sub-stream is created is $P = (1 - \frac{2}{|V|})^{2m}$.

Let $c = |E|/|V|$, then for the whole stream, the expectation of DF will be

$$E' = \sum_{m=0}^{|E|-1} (1 - \frac{2}{|V|})^{2m} = \frac{|V|^2}{4|V|-4}\big[1 - (1 - \frac{2}{|V|})^{2(|E|-1)}\big] \approx \frac{|V|}{4}\big(1 - e^{-4c}\big) \quad (4)$$

Algorithm 1. SWR sliding window randomization

```
1: Initalize $NumOfPartitions$, $AssignProportion$, $WinSize$, $WinMaxSize$ ...
2: while $|E| > 0$ do
3:     Fill $W$ with edges from $E$.
4:     Sort edges in $W$ by using the lower degree from two vertices in edge.
5:     Random.shuffe($W$, 0, $|W|/2$) and Random.shuffe($W$, $|W|/2$, $|W|$).
6:     for $i = 0 \to |W| * AssignProportion$ do        ▷ Start assigning edges
7:         $targetP \leftarrow SWRHeuristicRules(W[i])$
8:         $P_{targetP} \leftarrow P_{targetP} \cup \{W[i]\}$
9:     end for
10:    if $\frac{\max_{i\in n}\{|P_i|\} - \min_{i\in n}\{|P_i|\}}{\max_{i\in n}\{|P_i|\} + \min_{i\in n}\{|P_i|\}} > THRESHOLDUP$ then
11:        $WinSize \leftarrow min(MaxSize, WinSize * (1 + WindowSizeUpRate))$
12:    end if
13:    if $\frac{\max_{i\in n}\{|P_i|\} - \min_{i\in n}\{|P_i|\}}{\max_{i\in n}\{|P_i|\} + \min_{i\in n}\{|P_i|\}} < THRESHOLDDOWN$ then
14:        $WinSize \leftarrow WinSize/(1 + WindowSizeDownRate))$
15:    end if
16: end while
```

Generally $|E| > |V|$, therefore approximately we have $E' \approx \frac{|V|}{4}$. We have observed from various experiments that when the DF is order of magnitude higher than the target number of partitions, good balanced partitioning can be easily achieved with simple rule-based greedy algorithm. We don't single out the experimental result here due to space limit. And if a light-weight rule-based greedy algorithm can achieve good balance, then the heavier score calculating and additional imbalance adjustment will not be necessary any more. Our solution is a moving-window mechanism before the partitioning to reorder the incoming edges, so as to increase the observable DF. The basic idea is to keep a suitable window size allowed by the resource limit as long as the reordering (randomization) is sufficient to achieve good balance. For every round of assignment, the size of window will be adjusted according the balanced state of partitions.

Such mechanism can increase the DF effectively. Let us assume that the probability of a vertex with an earlier edge having same vertex is β. $\beta = \frac{2}{|V|}$ for a completely random edge stream. When $|E|$ is sufficiently large, the DF of the whole stream is $E' \approx \frac{1}{2\beta}$. When a window is used, assuming m edges have been processed, the probability that a vertex has appeared before is $\beta * \frac{m}{|E|}$, so at the n-th window, the probability that a vertex has appeared in front windows is $\beta * \frac{(n-1)*|W|}{|E|}$, and the expectation of the number of distinct vertices in the window is $|V|' = min(\frac{|W|}{c}, 2 + \frac{2}{\beta})$ so the expectation of the DF so far will be

$$E' = \sum_{n=1}^{\lceil \frac{|E|}{|W|} \rceil} \Big[\sum_{m=0}^{m=|W|-1} \big(1 - \beta * \frac{(n-1)*|W|}{|E|} - \frac{2}{|V|'}\big)^{2m} \Big] \tag{5}$$

For the natural graph datasets that resemble certain ordering patterns, the value of β is generally large, so the DF of the whole edge stream is largely dependent on the first several windows, especially the first one. Therefore, we advise to use bigger windows to achieve higher DF. If the buffer size provided to the moving window is limited, we propose to use the technique of *partial dispatch*, which only assigns a fraction of edges in the window and leave the rest to mix with new edges, so as to effectively increase the range of randomization. Furthermore, changing the edge order to distribute the edges related to the same vertex more evenly over time, the partial degree of a vertex observed so far will be a much more reliable indicator when used for relative comparison among vertices during the partitioning, while in a stream of heavy clustering effect the relative partial degree may fluctuate a lot as edges of different vertices will come in waves.

4.2 Getting Better Quality

In this subsection we will discuss the reordering mechanism inside the window and the heuristic rules to further improve the partitioning quality.

It has been shown that for graphs with power-law degree distribution, preferring to cut high-degree vertices during a graph partitioning is a good strategy to lower VRF [4,5]. So the cost of cutting a low-degree vertex is more expensive than a high-degree one and avoid cut low-degree vertex will be helpful. And streaming greedy GP means later partitioning decisions are greatly affected by the existing edge assignment. Therefore in the window we want to move edges with low-degree vertices upfront so they are less likely to be cut into different partitions. We divide edges into two equal parts for getting most randomization.

We start from the rules of greedy and extend them with windowed randomization and degree considerations to design the heuristic rules. DBH and HDRF have applied "preferring to cut high-degree vertices" into hashing and score-based greedy GP, and integrating it into a rule-based greedy GP should also be beneficial. Another useful observation is regarding to the large scale graphs from the social network. The principle of triadic closure tells us that the more common neighbors two separate vertices have the more likely a new edge will form between them. When processing graphs in stream, the arrival of edges can be seen as analog to the formation of a social network, so if a vertex has more neighbors in one partition, it is more likely that a new edge connecting its neighbors in that partition will exist down the stream, so we'd better assign a new neighbor to the partition with most neighbors. Similar edge clustering effect were observed also in graph datasets of other origins, so the principle is also applicable.

From the discussions above we define four assignment rules in our heuristics as below. More specifically, with partition set P and $A(v)$ as defined before and the partial degree record δ, for a new edge $e = (u, v)$, the rules are:

1. If none of u and v has been assigned before, put e to the smallest partition in P.
2. If one of u and v has been assigned before, let it be t, among the smallest M partitions in $A(t)$, put e to the one with the most neighbors.
3. If both u and v have been assigned before, and $B = A(u) \cap A(v) \neq \emptyset$, among the smallest M partitions in B, put e to the one with the most neighbors of both u and v.
4. Otherwise, let $t = (\delta(u) < \delta(v) \,?\, u \,|\, v)$, among the smallest M partitions in $A(t)$, put e to the one with the most neighbors. A new vertex replica is created accordingly.

4.3 Complexity Discussion and Summary

The complexity of rule-based algorithms is generally $O(|E|)$. `SWR` is a rule-based approach and the added pay is the reorder cost R which is unrelated to the number of target partitions, so the complexity is $O(|E|+R)$. For the score-based `HDRF`, for each edge assignment $|P|$ scores need to be calculated so the complexity is $O(|E| * |P|)$. `ADWISE` also uses a sliding-window but only to extend the score calculation to multiple edges in order to increase the range of local optima in the edge assigning process. While the clustering effect behind the imbalance problem is actually not specifically handled, being a score-based algorithm the complexity of `ADWISE` is further increased to $O(|E| * |P| * |W|)$.

The algorithm of `SWR` can be easily extend from single-machine to work in parallel across multiple machines to accelerate graph partitioning. Just like the case of `greedy` and `HDRF`, only some shared status information will be needed. Since we have proven that windowed randomization can increase the DF and a large DF makes balanced partitioning easy to achieve for simple greedy approach, with revised assigning rules, our `SWR` algorithm can achieve stable partitioning results that are good in all three aspects of quality, balance and speed.

5 Performance Evaluation

In this section we will present the experimental results about the performance of `SWR` in comparison with other representative GP algorithms: `hashing`, `DBH`, `grid`, `HDRF` and `ADWISE`. Comparisons will be carried out on multiple real-world and synthetic graph datasets with regard to VRF, balance and speed, respectively. In the end, we will also analyze how window size affects the balance and VRF performance in SWR with respect to different DF datasets.

Table 1. Datasets used in the experiments

Dataset	$\|V\|$	$\|E\|$	Intrinsic DF
BA-BFS	500000	12663357	1
BA-DFS	500000	12663357	1
BA-RANDOM	500000	12663357	$\gg$10000
as-skitter	1696415	11095298	1
soc-pokec-relationships	1632803	30622564	2
soc-LiveJournal	4847571	68993773	1042
higgs-twitter	456626	14855842	1408
Tecent Weibo	1944589	50635143	$\gg$10000
Wiki-Topcats	1791489	28511807	$\gg$10000

5.1 Experimental Settings and Datasets

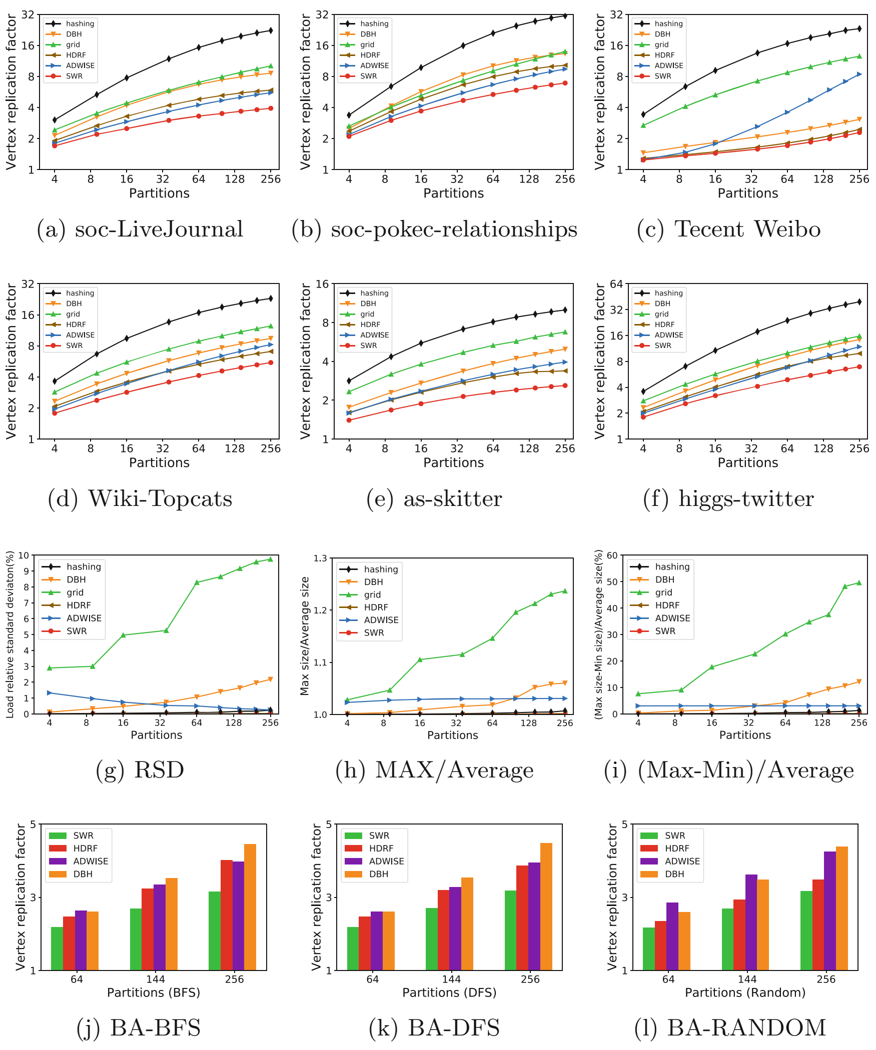
In our evaluations, three synthetic graph and six large real-world graph datasets were used to represent graphs with different DF corresponding to clustering effect from very strong, more general and rather weak as shown in Table 1 including the Tecent Weibo from KDD-Cup 2012. The others are from SNAP[1]. All the GP algorithms are implemented in Python except ADWISE, we use the source codes and parameters that are publicly available from the authors. For HDRF, we set the balancing factor $\lambda = 1.1$ as recommended by the authors and for ADWISE, the init window size is set to 100 and the same for SWR. The AssignProportion is set to 0.1 and the Neighbors is 3, meaning choosing from 3 smallest partitions with the most neighbors. Each experiment was repeated 10 times and the average was used as the result. The hardware platform is a workstation equipped with 12-core Intel Xeon CPUs and 64 GB physical memory.

5.2 Experimental Results and Discussions

Performance on Partitioning Quality, Balance and Speed

VRF: NumOfPartitions is set from 4 to 256, and the results are shown in Fig. 4(a)–(f). `SWR` is always the best performer. In particular, `DBH` always performs better than `hashing` with the only additional strategy on cutting high degree vertex first, showing the universal effectiveness of such strategy, which is especially strong on the Tencent Weibo dataset. However, `ADWISE` performs bad on this dataset although it is as good as `HDRF` on other datasets, which means that by considering more edges during the score calculation, the power-law degree distribution is not well handled. The performance of other algorithms is more consistent among different datasets. `SWR` makes better use on both the degree information as well as the neighboring information, so it achieves the overall best performance on all the tested datasets, which is 15% to 30% on

[1] http://snap.stanford.edu.



(a) soc-LiveJournal (b) soc-pokec-relationships (c) Tecent Weibo

(d) Wiki-Topcats (e) as-skitter (f) higgs-twitter

(g) RSD (h) MAX/Average (i) (Max-Min)/Average

(j) BA-BFS (k) BA-DFS (l) BA-RANDOM

Fig. 4. VRF/Balance against given number of partitions.

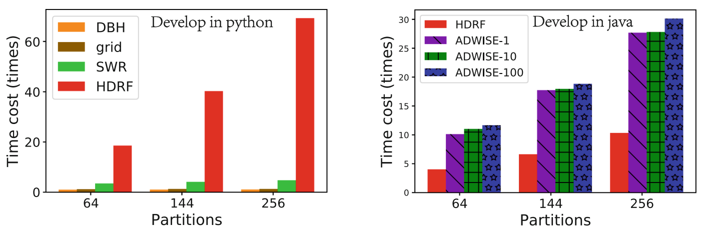
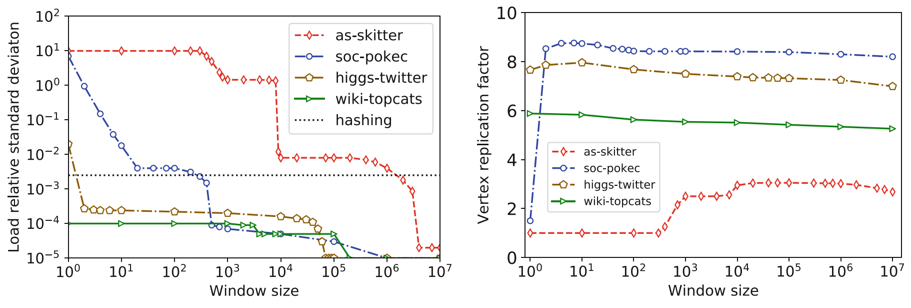


Fig. 5. Partitioning time against number of partitions. Time cost represents the times compared to `hashing`. For `ADWISE`-X, X represents the init window size.



(a) RSD (b) VRF

Fig. 6. Balance/VRF against different window sizes.

average better than `HDRF` or `ADWISE`. Figure 4(j)–(l) show the VRF of different algorithms on the same synthetic graph generated by BA model but collected by BFS, DFS and Random. `HDRF` performs the best on random graph, confirming the previous discussion of edge correlation influencing its performance. `SWR` outperforms `HDRF` even on random graph, because it not only solves the problem of edge correlation, but also takes better use of the vertex degree and neighboring information.

Balance: The experiments are carried out on Tecent Weibo dataset. Three measures were used to show how the algorithms perform from different angles. RSD represents the deviation of the partition size. Max, Min and Average represent the maximal, minimal and average size of all the partitions. The results are shown in Fig. 4(g)–(i). We can see that `SWR` and `HDRF` provide the best performance. The balance of `hashing` is a bit worse for purely depending on probability. `DBH` and `ADWISE` can only provide rather moderate balance. The balance performance of `ADWISE` is special: when the target number of partitions increases, RSD decreases but MAX/Average and (MAX-MIN)/Average keep constant. `grid` performs bad as the number of partitions grows, the imbalance increases fast until the partitioning result becomes unacceptable.

Speed: We compare the partitioning time of all the algorithms on Tecent Weibo datasets with 64, 144 and 256 partitions and the result is shown in Fig. 5. Since `ADWISE` was tested with the original source code implemented in Java while all the others were implemented in Python, we can not comparing them directly. Thankfully `hashing` and `HDRF` are implemented in both environment, so we use `hashing` as the reference to normalize the time to rule out the differences of environment. `HDRF` is used as the anchor when we compare `ADWISE` with the other algorithms we implemented. We can find `SWR` has the good feature as `grid` and `DBH`, whose time does not grow with the number of partitions, while `HDRF` and `ADWISE` need much more time which also grows linearly with the number of partitions. The differences of `HDRF` between Fig. 5(a) and (b) is due to the different implementation. `ADWISE` with different init window size behave similar as the window size will always adjust to 1000 as configured in the system. To our surprise, `ADWISE` is not too much slower than `HDRF`, for it uses a Lazy Window Traversal strategy, but they are still slower than all the rule-based algorithms.

Performance Sensitivity to Window Size

Finally we analyze how much the window size affects `SWR` performance on different DF datasets. In order to make the results reliable, we give a different window size for each experiment and don't allow it to adjust automatically. The AssignProportion is set to 1, Neighbors to 0, and NumOfPartitions to 100 and just randomizing all edges in the window. As the RSD of `hashing` is stable on different datasets we use it as reference. Since the DF of as-skitter dataset is 1, it's more difficult to achieve good balanced partitioning on it. As we can see in Fig. 6(a), when the window size is less than 1000, the windowed mechanism has no effect. This can be confirmed by the VRF of 1 in Fig. 6(b), which means that all the edges are assigned to the same partition. When the window size reaches 1000 and beyond, the windowed mechanism starts to help break the self-constrained edge stream to several sub-streams, there will be more but unbalanced partitions, so the VRF grows. When the window size reaches 10000, a sharp change happens and the RSD is in the level of 0.01 which is as good as `hashing`. All partitions get an acceptable balance and VRF become stable. This indicates that a window size of 10000 is generally sufficient for balanced partitioning and it is practically easy to release. As the window size keeps growing, both RSD and VRF declines, though smaller and smaller. This is because with larger window, randomization covers a larger range to break the clustering effect, and the degree and neighboring information also become more reliable. As the window size becomes very big, RSD and VRF converge to values that depend on the dataset itself. For soc-pokec-relationships dataset, a much small window size can make it balanced, it just needs a window of 100 to reach the balance level of `hashing`, although its DF is 2. A very small window can initialize a good balanced partitioning with `SWR` on it. The same phenomenon is observed on higgs-twitter dataset as well, it has a DF of 1408 and a window of 3 is enough for it to keep balance in many cases. As for wiki-topcats dataset, it has a very large DF, so it is balanced naturally with RSD below `hashing`. This indicates `SWR` can get better balance than `hashing`. Although maybe a small window can help to keep balance, we suggest a big init window size for lower VRF and RSD despite the window will change itself.

6 Conclusions

We found that the intrinsic edge order of natural graphs makes it difficult for existing graph partitioning algorithms to achieve balanced partitioning results. The reason is the clustering effect embedded in the intrinsic order. We propose a sliding window mechanism to reorder the edges, making it much easier for fast rule-based greedy approach to achieve balance. Considerations on vertex degree and neighboring information further improves the partitioning quality. Experimental results show that the proposed `SWR` algorithm achieves good performance in all aspects of quality, balance and speed, and outperforms all competing algorithms. Comparing to the state-of-the-art `HDRF`, partitioning speed is increased by 3–20 times, and quality is improved by 15% to 30% on average.

References

1. Abou-Rjeili, A., Karypis, G.: Multilevel algorithms for partitioning power-law graphs. In: Proceedings 20th IEEE International Parallel & Distributed Processing Symposium, 10 pp. IEEE (2006)
2. Andreev, K., Racke, H.: Balanced graph partitioning. Theory Comput. Syst. **39**(6), 929–939 (2006)
3. Chen, R., Shi, J., Chen, Y., Zang, B., Guan, H., Chen, H.: PowerLyra: differentiated graph computation and partitioning on skewed graphs. ACM Trans. Parallel Comput. (TOPC) **5**, 13 (2019)
4. Cohen, R., Erez, K., Ben-Avraham, D., Havlin, S.: Resilience of the internet to random breakdowns. Phys. Rev. Lett. **85**(21), 4626 (2000)
5. Cohen, R., Erez, K., Ben-Avraham, D., Havlin, S.: Breakdown of the internet under intentional attack. Phys. Rev. Lett. **86**(16), 3682 (2001)
6. Crucitti, P., Latora, V., Marchiori, M., Rapisarda, A.: Error and attack tolerance of complex networks. Phys. A **340**(1–3), 388–394 (2004)
7. Feige, U., Hajiaghayi, M., Lee, J.R.: Improved approximation algorithms for minimum weight vertex separators. SIAM J. Comput. **38**(2), 629–657 (2008)
8. Gonzalez, J.E., Low, Y., Gu, H., Bickson, D., Guestrin, C.: PowerGraph: distributed graph-parallel computation on natural graphs. In: Presented as Part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2012), pp. 17–30 (2012)
9. Gonzalez, J.E., Xin, R.S., Dave, A., Crankshaw, D., Franklin, M.J., Stoica, I.: GraphX: graph processing in a distributed dataflow framework. In: 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2014), pp. 599–613 (2014)
10. Halberstam, H., Laxton, R.R.: Perfect difference sets. Glasgow Math. J. **6**(4), 177–184 (1964)
11. Jain, N., Liao, G., Willke, T.L.: GraphBuilder: scalable graph ETL framework. In: First International Workshop on Graph Data Management Experiences and Systems, pp. 1–6. ACM (2013)
12. Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM J. Sci. Comput. **20**(1), 359–392 (1998)
13. Kim, M., Candan, K.S.: SBV-Cut: vertex-cut based graph partitioning using structural balance vertices. Data Knowl. Eng. **72**, 285–303 (2012)
14. Leskovec, J., Lang, K.J., Dasgupta, A., Mahoney, M.W.: Community structure in large networks: natural cluster sizes and the absence of large well-defined clusters. Internet Math. **6**(1), 29–123 (2009)
15. Low, Y., Gonzalez, J., Kyrola, A., Bickson, D., Guestrin, C.: GraphLab: a distributed framework for machine learning in the cloud. arXiv preprint arXiv:1107.0922 (2011)
16. Malewicz, G., et al.: Pregel: a system for large-scale graph processing. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, pp. 135–146. ACM (2010)

17. Petroni, F., Querzoni, L., Daudjee, K., Kamali, S., Iacoboni, G.: HDRF: stream-based partitioning for power-law graphs. In: Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, pp. 243–252. ACM (2015)
18. Rahimian, F., Payberah, A.H., Girdzijauskas, S., Haridi, S.: Distributed vertex-cut partitioning. In: Magoutis, K., Pietzuch, P. (eds.) DAIS 2014. LNCS, vol. 8460, pp. 186–200. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43352-2_15
19. Stanton, I., Kliot, G.: Streaming graph partitioning for large distributed graphs. In: Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1222–1230. ACM (2012)
20. Xie, C., Yan, L., Li, W.J., Zhang, Z.: Distributed power-law graph computing: theoretical and empirical analysis. In: Advances in Neural Information Processing Systems, pp. 1673–1681 (2014)
21. Zhang, C., Wei, F., Liu, Q., Tang, Z.G., Li, Z.: Graph edge partitioning via neighborhood heuristic. In: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 605–614. ACM (2017)