Quasi-Streaming Graph Partitioning: A Game Theoretical Approach

Qiang-Sheng Hua, *Member, IEEE*, Yangyang Li, Dongxiao Yu, *Member, IEEE*, and Hai Jin, *Fellow, IEEE*

Abstract—Graph partitioning is a fundamental problem to enable scalable graph computation on large graphs. Existing partitioning models are either streaming based or offline based. In the streaming model, the current edge needs all previous edges' partition choices to make a decision. As a result, it is hard to carry out partitioning in parallel. Besides, offline based partitioning requires full knowledge about the input graph which may not suit well for large graphs. In this work, we propose a quasi-streaming partitioning model and a game theory based solution for the edge partitioning problem. Specifically, we separate the whole edge stream into a series of batches where the batch size is a constant multiple of the number of partitions. In each batch, we model the graph edge partitioning problem as a game process, where the edge's partition choice is regarded as a rational strategy choice of the player in the game. As a result, the edge partitioning problem is decomposed into finding Nash Equilibriums in a series of game processes. We mathematically prove the existence of Nash Equilibrium in such a game process, and analyze the number of rounds needed to converge into a Nash Equilibrium. We further measure the quality of these Nash Equilibriums via computing the PoA (Price of Anarchy), which is bounded by the number of partitions. Then we evaluate the performance of our strategy via comprehensive experiments on both real-world graphs and random graphs. Results show that our solution achieves significant improvements on load balance and replication factor when compared with five exsiting streaming partitioning strategies.

Index Terms—Graph edge partitioning, exact potential game, Nash Equilibrium, replication factor, load balance

1 Introduction

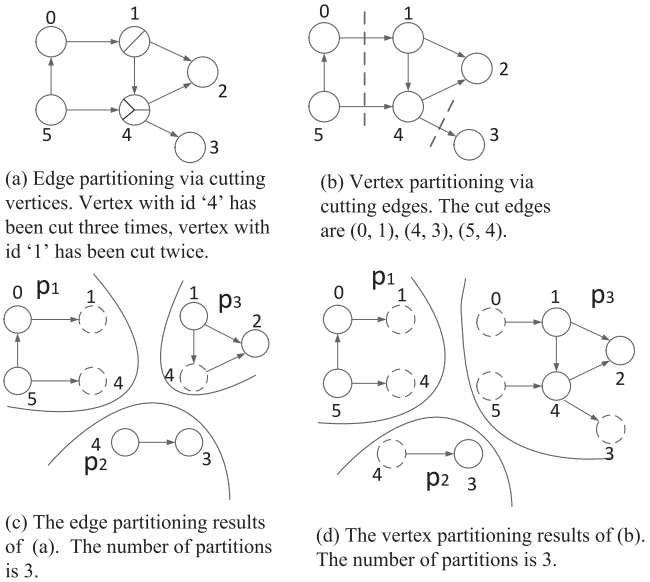
Nowadays big graphs have emerged in a wide range of real applications, such as web graphs and online social networks. Mining knowledge from these graphs faces a great challenge since their volume is explosively increasing. To perform graph computing on distributed systems with several machines, partitioning the entire graph across machines is an imperative procedure. Graph partitioning can significantly affect the performance of distributed graph computing systems in terms of communication cost and workload balance. Existing graph partitioning strategies can be classified into two categories: vertex partitioning and edge partitioning. Vertex partitioning leads to vertices being disjoint in different partitions, and edges being spanned between two partitions. On the contrary, edge partitioning leads to edges being separated in different partitions, and vertices being spanned among several partitions. The difference between vertex partitioning and edge partitioning is illustrated in Fig. 1.

- *Q.-S. Hua, Y. Li, and H. Jin are with the Services Computing Technology and System Lab, Big Data Technology and System Lab, Cluster and Grid Computing Lab, School of Computer Science and Technology, Huazhong University of Science and Technology, 1037 Luoyu Road, Wuhan 430074, P.R. China. E-mail: {qshua, yangyli, hjin}@hust.edu.cn.*
- *D. Yu is with the School of Computer Science and Technology, Shandong University, Qingdao 266237, P.R. China. E-mail: dxyu@sdu.edu.cn.*

Manuscript received 3 June 2018; revised 19 Dec. 2018; accepted 21 Dec. 2018. Date of publication 1 Jan. 2019; date of current version 12 June 2019. (Corresponding author: Dongxiao Yu.)
Recommended for acceptance by Y. Yang.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TPDS.2018.2890515

In the vertex partitioning scheme, the workload of each partition is determined by the number of vertices hosted in it. The communication cost is determined by the number of edges crossing different partitions. In the edge partitioning scheme, the workload of each partition is determined by the number of edges hosted in it. Besides, the communication cost is determined by the total number of replicas of all vertices. The vertex partitioning scheme has been employed in most distributed graph computing frameworks such as Pregel [15], GraphLab [14] and Giraph [19].

Most real-world graphs follow the power law degree distribution. It has been proved that edge partitioning scheme can achieve better performance for power law graphs [9]. For this reason, several edge partitioning based strategies have been proposed such as DBH [24] and HDRF [18]. However, DBH [24] makes use of global degree information to always cut the high degree vertices. It might be infeasible in the real streaming setting, since in streaming processing systems such as [22], the computation is performed continuously and uninterruptedly. As for HDRF [18], it uses the partial degree to help make decisions. It maintains a global degree table with partitial degree of each vertex. When a new edge arrives, the degrees of two endpoints are updated. However, these solutions may not suit well in real streaming setting either. In real streaming setting, it is not possible to compute global degrees information in advance. Moreover, storing the global degree table entails the $\Theta(m)$ memory consumption where m is the number of edges of a graph. For a commodity machine, this might be unacceptable when the number of edges of a graph exceeds a limit. Moreover, it may also be inefficient to perform lookup in such a huge table.



(a) Edge partitioning via cutting vertices. Vertex with id '4' has been cut three times, vertex with id '1' has been cut twice.

(b) Vertex partitioning via cutting edges. The cut edges are (0, 1), (4, 3), (5, 4).

(c) The edge partitioning results of (a). The number of partitions is 3.

(d) The vertex partitioning results of (b). The number of partitions is 3.

Fig. 1. (a) Edge partitioning and (b) vertex partitioning. (c) and (d) are the partitioning results of (a) and (b) respectively. Vertices with dashed circles are replicas.

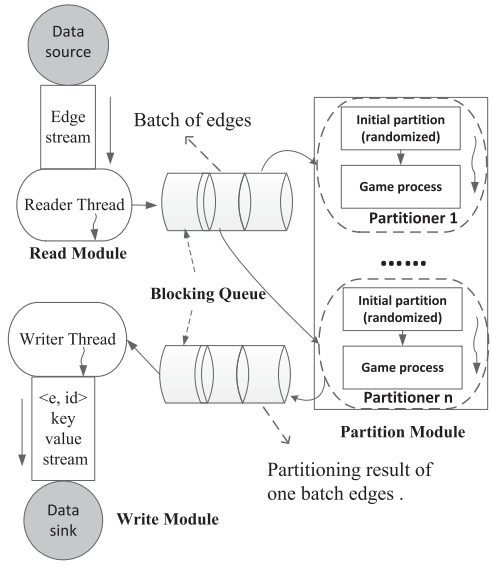


Fig. 2. The quasi-streaming model. Each partitioner is wrapped into a thread. The Reader thread reads a batch of edges from disk to memory each time, then the batch of edges are fed into the partition module. All partitioners run in parallel within the partition module. The partitioning results are represented as key value pairs, where the key is an edge e and the value is the partition id for the edge e. These results are saved in the data sink.

In addition, existing solutions based on streaming model lack of parallelism (more details refer to Section 3.1). For example, Stanton and Kliot [21] regard performing streaming graph partitioning in parallel as their future work, quoted as "*The second direction is to address using parallel loaders. ...We expect the performance of running parallel loader on independent portions of the graph stream...* ".

Based on these observations, in this paper we propose a quasi-streaming model and a novel corresponding edge partitioning strategy derived from game theory, which is demonstrated as Fig. 2. In this model, the whole edge stream is separated into series of batches, where the batch size (some constant number of edges) is an input parameter. In each batch, all edges constitute the players set in a game process. The edge's partition choice is regarded as the rational strategy in the game. When the game process of each batch reaches a Nash Equilibrium, i.e., no edge has incentive to unilaterally change its current partition choice, the partitioning task of this batch is finished. The partitioning task of each batch can be accomplished by a partitioning thread, which corresponds to a partitioner in Fig. 2. All partitioning threads can carry on in parallel. As mentioned before, the quasi-streaming model might be an interesting attempt to discover potential parallelism from the streaming model [21], where similar performance is expected via partitioning independent portions of the graph stream in parallel.

Compared with the streaming model and the offline model, the total memory consumption of our quasi-streaming model is only $\Theta(batch\_size * number\_of\_threads)$, which is much smaller than the $\Theta(m)$ memory consumption in both the streaming model and the offline model. Furthermore, in contrast with the streaming model, the time used by the quasi-streaming model could be also greatly reduced due to the introduced parallelism. These theoretical analyses are also corroborated in the experiment (Section 6.2).

Notice that most real world graphs are collected by crawling in breadth-first manner [3], [25]. As a result, when the graph is stored as the edge list form, edges incident on the same vertex are stored next to each other. In addition, most real-world graphs' degrees follow the power law distribution, where most vertices have small degrees. If the batch size is larger than most vertices' degrees, we can regard these subgraphs composed by each batch approximately disjoint. As a result, we can perform partitioning on these batches in parallel. We need to point out that if the above assumption does not hold, i.e., if the incident edges of a vertex are not present consecutively in the input, we can perform a preprocessing step where more details will be given in Section 6.2.

Note that it seems like the hypothesis of the quasi-streaming model is much stronger than the streaming model, since an edge can deviate its current partition choice according to other edges' choices within the same batch (more details refer to Section 5) in the quasi-streaming model. However, the batch size in the quasi-streaming model is a constant multiple of the number of partitions. As a result, the information within each batch is quite limited. In contrast, in the streaming model each edge can make use of all previous edges' partition choices via a one-pass manner. Hence, performing efficient partitioning in the quasi-streaming could be much harder than in the streaming model.

We list our contributions as follows:

- We propose the quasi-streaming model for edge partitioning problem and a novel partitioning strategy based on game theory for this model.
- We mathematically prove the existence of pure Nash Equilibrium in the quasi-streaming edge partitioning game and analyze the number of rounds needed to converge into a Nash Equilibrium. We prove the *Price of Anarchy* PoA, i.e., the ratio between the worst Nash Equlibrium and social optimum, is bounded by the number of partitions in our edge partitioning game.

- The performance of our method is evaluated via comprehensive experiments on both large real-world graphs and random graphs.

The rest of our paper is structured as follows. Section 2 introduces the related work. System model and problem definition are shown in Section 3. We present the quasi-streaming edge partitioning game with theoretical analyses in Section 4. Section 5 shows a *Best Response Dynamics* [17] based algorithm to find a Nash Equilibrium. Section 6 evaluates the performance of our strategy. Section 7 concludes the work.

2 Related Work

Graph partitioning plays a key role in parallel and distributed graph computing applications, since data layout has a significant impact on these applications' performance in terms of communication cost and workload balance. Due to the intrinsic hardness [1], [8] of graph partitioning, many heuristic partitioning strategies have been proposed in practice. The state of the art heuristic partitioning strategies can be divided into offline based and streaming based. METIS [20] is a well-regarded offline heuristic, which combines a quantity of heuristics such as the well-known Kergnighan-Lin heuristic [12]. However, offline based heuristics [11] require full knowledge about the graph, which might be impractical for large-scale graphs and dynamic graphs. A hybrid heuristic is proposed in Powerlyra [7], which combines vertex partitioning and edge partitioning schemes together. The hybrid heuristic is called Ginger, where an extra reassignment phase is needed for the high degree vertices after the original partitioning.

Balanced graph partitioning problem in streaming setting was first introduced by Stanton [21]. Several greedy strategies are proposed in [21], which assign current vertex to the partition based on load balance and the number of neighbors hosted in each partition. Fennel [23], another vertex partitioning scheme, combines two heuristics together: placing the newly arrived vertex in the partition holding the most number of neighbors or holding the least number of non-neighbors. Note that, the authors in [5] implemented an edge partitioning version of Fennel. A game theory based solution for the specific vertex partitioning problem is proposed in [2], which is quite different from the conventional streaming graph vertex partitioning [21] in terms of both settings and optimization objectives.

The heuristic for streaming edge partitioning was first proposed in [9], where the edge partition based scheme has been proved more efficient than vertex partitioning based scheme for power law graphs [9] . The other two streaming edge partitioning schemes, DBH [24] and HDRF [18] make use of the skewed degree distribution characteristics for natural graphs. A survey for the graph partitioning heuristics can be found in [6].

3 System Model and Problem Definition

3.1 System Model

Before introducing the quasi-streaming partitioning model, we briefly review the intensively studied streaming partitioning model [18], [21]. For the streaming vertex partitioning problem, vertices arrive with the set of their neighbors. For the streaming edge partitioning problem, each edge arrives with its two endpoints. In these two variants, the heuristics decide to place the current vertex or edge on a certain partition based the all previous vertices' or edges' partition choices, which is implemented via maintaining a global table recording all previous vertices' or edges' partition choices. Moreover, for degree based heuristics such as DBH [24] and HDRF [18] in the edge partitioning problem, an extra degree table has to be maintained to record the degree information of each vertex.

For the real streaming scenario, the stream may be extremely huge or even infinite [22], thus these solutions may be inefficient in terms of huge table's memory usage and inefficient lookup on such a huge table. Besides , it is hardly to perform partitioning in parallel, since the global tables are read and written intensively. In order to make a partition choice, each edge has to obtain the number of neighbors hosted in each partition and the load of each partition before making the choice. When the edge has made its choice, it has to record its choice in these tables to benefit the successors in the stream.

Based on these two observations, we propose the quasi-streaming model aimed at achieving more parallelism and less memory usage. We formally define the quasi-streaming partition model for edge partitioning problem as follows:

- The whole graph is represented by an edge stream, where each edge arrives with its endpoints.
- The whole stream is segmented as a series of batches, where the batch size is specified as an input parameter, which is a constant multiple of the number of partitions. The number of partitions k is an input parameter as well.
- In each batch, the partitioning problem is modeled as a game process. When a Nash Equalibrium is found, the partitioning of this batch is finished. More details are given in Sections 3.3 and 4.
- The Nash Equibrium is found via a *Best Response Dynamics* based algorithm. More details refer to Section 5.

In the quasi-streaming model, the current edge can take full advantage of other edges' partition choices within the same batch via an iterative fashion, which can achieve a better local optimum compared with the streaming based solutions. There is no overhead to maintain a huge gobal table to record the all previous edges' choices. We only use a local table to record other edges's choices within the same batch, which is far smaller than the global one. Moreover, the local table's memory can be reused when the current batch has finished its partitioning.

3.2 Strategic Game Theory

In order to model the edge partitioning problem in each batch as a game, we introduce some basic game theory concepts.

In strategic games, players are assumed as selfish and rational to optimize their individual objective (or cost) functions. Based on this assumption, the player chooses a strategy minimizing its own cost without considering the effect of its choice on other players' objectives. Formally, a strategic game is defined as:

- The set of players $\mathcal{N} = \{1, \ldots, N\}$.
- The strategy space $\mathbb{S} = \mathbf{S_1} \times \ldots \times \mathbf{S_N}$, where $\mathbf{S_i}$ is the strategy set of player i, i.e., all actions where player i

TABLE 1
Notations

Notations	Description
$G_b = (V_b, E_b)$	the subgraph stream in batch b
$G = (V, E)$	$\bigcup_b G_b$
B	batch size
p_i	denotes the partition with ID p_i
P	the set of k partitions $\{p_1, \ldots, p_k\}$
$L(p_i)$	the number of edges in $\{e \mid e \in E, e \text{ locates in } p_i\}$
$l(p_i)$	the number of edges in $\{e \mid e \in E_b, e \text{ locates in } p_i\}$
$d(p_i, v)$	the number of edges incident on $v \in V_b$ in p_i
S_e	the partition choice of edge $e \in E_b$
S_{-e}	the partition choicies of all edges except e
$l(S_e)$	the number of edges hosted in partition S_e
$Rep(v)$	the set of partitions containing v's replica
$C_e(S_e, S_{-e})$	individual cost function of edge e
$C(S_e, S_{-e})$	the social welfare function
σ	the standard deviation of edge load
rep	the replication factor

can select from. $S = (S_1, \ldots, S_N) \in \mathbb{S}$ is called a strategy profile, and $S = (S_i, S_{-i})$ is its abbreviation form, where S_{-i} denotes all players' strategy choices except player i.

- The objective (cost) function $C_i(S) : \mathbb{S} \mapsto \mathbb{R}$ for each player i, i.e., the objective the player tries to minimize or maximize.

A strategic game is determined via the tuple $\mathcal{G} = \{\mathcal{N}, \mathbb{S}, \{C_i\}_{i \in \mathcal{N}}\}$. The social welfare is usually defined as $\sum_{i \in \mathcal{N}} C_i(S)$, which we want to minimize.

Definition 1. *A* pure strategy *provides a complete definition of how a player will play a game for any situation.*

Definition 2. *The pure strategy profile* $S^* \in \mathbb{S}$ *is* pure strategy Nash Equilibrium *if and only if for each* $i \in \mathcal{N}$*:*

$$C_i(S_i{}^*, S^*_{-i}) \leq C_i(S'_i, S^*_{-i}), \tag{1}$$

where $S_i{}^* \neq S'_i, S'_i \in \mathbf{S_i}$ is another strategy. In other words, no players have incentives to deviate from their current strategies unilaterally.

3.3 Problem Definition

Now we define the edge partitioning problem in the quasi-streaming model. We use $G_b = (V_b, E_b)$ to represent the subgraph in each batch b and $G = (V, E)$ to represent the whole graph stream consisting of all batches. Given the number of partitions k, our goal is to assign edges to partitions evenly, while minimizing the average replicated times of each vertex. We follow the same balanced edge partitioning definition used in [9], [18], [24], formulated as follows:

$$\begin{aligned} &\min \frac{1}{|V|} \sum_{v \in V} |Rep(v)| \\ &\text{s.t.} \max_{p_i \in P} L(p_i) \ < \ \lambda \frac{|E|}{k}, \end{aligned} \tag{2}$$

where $Rep(v) \subseteq P$ is the set of partitions which hold the replica of vertex v and $\lambda \geq 1$ is a small constant. $P = \{p_1, \ldots, p_k\}$ is the set of partitions, and $L(p_i)$ represents the number of edges hosted in partition p_i in terms of all batches, i.e., $\sum_{i=1}^{k} L(p_i) = |E|$. Note that the edge partitioning problem is NP-hard [1], [8].

The partitioning quality in this paper is measured via two metrics similar to [9], [18]:

- *σ (Standard deviation of edge load)*: the standard deviation of the number of edges hosted in each partition.
- *rep (Replication factor)*: the average replicated times of each vertex, defined as $\frac{1}{|V|}\sum_{v \in V} |Rep(v)|$, where $Rep(v)$ is the set of partitions which hold the replica of vertex v.

The *Replication factor* metric is the same as [9], [18]. But our load balance metric σ is different from the *Relative Standard deviation* metric which is defined as the ratio of σ over the average load in [18]. The σ metric is different from the metric defined as the ratio of the max load over the average load in [23], [24] as well. Note that our standard of deviation metric σ is a much harsher metric than these two metrics for measuring the load balance status, which may distinguish partitioning strategies further in terms load balance.

4 Quasi-Streaming Edge Partition Game

In order to model the edge partitioning problem as a game process, we regard each edge as a player, and edges' partition choices are equivalent to players' strategy choices. In game processes, when a Nash Equilibrium is found, it indicates that we have arrived at a local optimum of social welfare objective. So we want to explore whether we can find a good local optimum of edge partitioning problem via building a bridge between the objective function in edge partitioning problem and social welfare function in the game process. In this section, we first build a game process related to the edge partitioning problem. Then we demonstrate that the game process can always converge into a Nash Equilibrium. Finally we measure the quality of these Nash Equilibriums via PoA (*Price of Anarchy*). The main notations and their descriptions used in this paper are listed in Table 1.

4.1 Game Construction

First of all, we define the edge partitioning objective function *QGEP* (Quasi-streaming Graph Edge Partitioning) in each batch as follows:

$$QGEP(G_b, P) = \alpha\beta\sigma^2 + (1 - \alpha) \sum_{v \in V_b} |Rep(v)| \tag{3}$$

where G_b is denoted as the subgraph in each batch and P is a set of partitions. σ is the standard deviation of edge load in each partition mentioned in Section 3.3. $\alpha \in (0, 1)$ is the preference factor between these two metrics. β is a normalization factor, which will be discussed elaborately in Section 4.2. According to the *QGEP* objective function, we define the social welfare function of edge partitioning game as follows:

$$C(S) = \alpha\beta \frac{1}{k} \sum_{i=1}^{k} l^2(p_i) + (1 - \alpha) \sum_{v \in V_b} |Rep(v)| \tag{4}$$

where $S = (S_1, \ldots, S_{|E_b|})$ is a strategy profile, which is defined in Section 3.2. $l(p_i)$ is the number of edges hosted in partition p_i in terms of each batch. Since each batch is

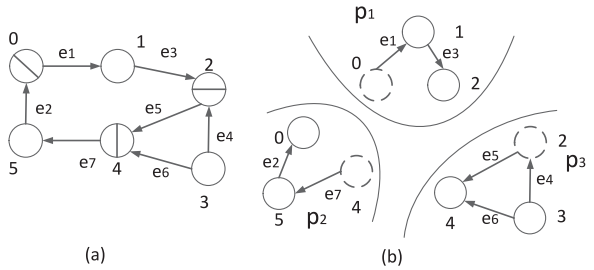


Fig. 3. The partitioning result of (a) based on Algorithm 2 is shown on (b). Vertices with dashed circles are replicas. Table 2 shows computation in details.

partitioned separately, we have $\sum_{i=1}^{k} l(p_i) = |E_b|$. Now we will show the equivalence between $C(S)$ and *QGEP*.

Proposition 1. *The difference between social welfare $C(S)$ and* QGEP*'s objective function is a constant value.*

Proof. Since the second part in $C(S)$ and *QGEP* is same , we only consider the first part. We first denote $a = \frac{|E_b|}{k}$ which is a constant since $|E_b|$ is a multiple constant of k. Note that $\sum_{i=1}^{k} l(p_i) = |E_b|$, then we have:

$$\begin{aligned}\sigma^2 &= \frac{1}{k}\sum_{i=1}^{k}(l(p_i)-a)^2 \\ &= \frac{1}{k}\left[\sum_{i=1}^{k} l^2(p_i) - 2a\sum_{i=1}^{k} l(p_i) + ka^2\right] \\ &= \frac{1}{k}\sum_{i=1}^{k} l^2(p_i) - \frac{1}{k}2a|E_b| + a^2 = \frac{1}{k}\sum_{i=1}^{k} l^2(p_i) - a^2.\end{aligned} \tag{5}$$

Therefore, $QGEP = \alpha\beta\sigma^2 + (1-\alpha)\sum_{v\in V_b}|Rep(v)| = C(S) - \alpha\beta a^2$. □

Next, we define the individual cost function for each player based on social welfare $C(S)$. For each edge e, the cost of e choosing partition S_e is given by:

$$C_e(S) = \alpha\beta\frac{1}{k}l(S_e) + (1-\alpha)\left(\frac{1}{d(S_e,u)} + \frac{1}{d(S_e,v)}\right), \tag{6}$$

where e is incident on vertex u and v, $d(S_e,u)$ is the number of edges which are hosted in partition S_e and incident on vertex u.

Proposition 2. *The social welfare $C(S)$ is the sum of all edges' individual costs in this game.*

Proof. First, we denote $E(p_i) = \{e | e \text{ hosted in } p_i, e \in E_b\}$ as the set of edges hosted in partition p_i and $incident(p_i, v) = \{e | e \text{ incident on vertex } v, e \in E(p_i)\}$. $\mathbf{1}\{condition\}$ is the indicator function, which equals to 1 when *condition* is true otherwise 0. $|E(p_i)| = l(p_i)$, $|incident(p_i,v)| = d(p_i,v)$. Then we have:

$$\begin{aligned}&\sum_{e\in E_b} C_e(S) \\ &= \alpha\beta\frac{1}{k}\sum_{e\in E_b} l(S_e) + (1-\alpha)\sum_{e\in E_b}\left[\frac{1}{d(S_e,u)} + \frac{1}{d(S_e,v)}\right] \\ &= \alpha\beta\frac{1}{k}\sum_{i=1}^{k}\sum_{e\in E(p_i)} l(p_i) + (1-\alpha)\sum_{i=1}^{k}\sum_{S_e=p_i}\left[\frac{1}{d(S_e,u)} + \frac{1}{d(S_e,v)}\right] \\ &= \alpha\beta\frac{1}{k}\sum_{i=1}^{k} l^2(p_i) + (1-\alpha)\sum_{i=1}^{k}\sum_{\substack{w\in V_b \\ p_i\in Rep(w)}}\frac{|incident(p_i,w)|}{d(p_i,w)} \\ &= \alpha\beta\frac{1}{k}\sum_{i=1}^{k} l^2(p_i) + (1-\alpha)\sum_{i=1}^{k}\sum_{w\in V_b}\mathbf{1}\{p_i\in Rep(w)\} \\ &= \alpha\beta\frac{1}{k}\sum_{i=1}^{k} l^2(p_i) + (1-\alpha)\sum_{w\in V_b}|Rep(w)| = C(S).\end{aligned} \tag{7}$$

□

Specifically, in Fig. 3, $\sum_{e\in E_b}[\frac{1}{d(S_e,u)} + \frac{1}{d(S_e,v)}] = \underbrace{[(\frac{1}{1}+\frac{1}{2}) + (\frac{1}{1}+\frac{1}{2})]}_{p_1:e_1,e_3} + \underbrace{[(\frac{1}{1}+\frac{1}{2}) + (\frac{1}{1}+\frac{1}{2})]}_{p_2:e_2,e_7} + \underbrace{[(\frac{1}{2}+\frac{1}{2}) + (\frac{1}{2}+\frac{1}{2}) + (\frac{1}{2}+\frac{1}{2})]}_{p_3:e_4,e_5,e_6} = \sum_{v\in V_b}|Rep(v)| = 9$. We denote the game constructed here as *EPG* (Edge Partitioning Game), i.e., $EPG = \{E_b, \mathbb{S}, \{C_e\}_{e\in E_b}\}$, where $\mathbb{S} = \{1,\ldots,k\}\times\ldots\times\{1,\ldots,k\}$ is the Cartesian product of all edges' strategy sets. In the rest of paper, we use $EPG = \{E_b, \mathbb{S}, \{C_e\}\}$ for brevity.

4.2 Normalization Issue

Note that, without normalization parameter β, the value of load balance part $C^{BAL}(S) = \alpha\frac{1}{k}\sum_{i=1}^{k} l^2(p_i)$ is not comparable to the value of replication factor part $C^{REP}(S) = (1-\alpha)\sum_{v\in V_b}|Rep(v)|$ in the social welfare function (Eq. (4)). The load balance part may be far larger than the replication part in the social welfare function without the normalization

TABLE 2
Algorithm 2 on Fig. 3, $k = 3, \alpha = 0.5, \beta = 1.6$

Steps		p_1	p_2	p_3
	Initialization	$e_1(1.235), e_2(1.485), e_3(1.485)$	$e_7(1.245)$	$e_4(1.235), e_5(1.235), e_6(1.235)$
Round 1	$e_1(\underline{1.235}, 1.490, 1.980)$	$e_1(1.235), e_2(1.485), e_3(1.485)$	$e_7(1.245)$	$e_4(1.235), e_5(1.235), e_6(1.235)$
	$e_2(1.485, \underline{1.240}, 1.980)$	$e_1(1.235), e_3(1.485)$	$\mathbf{e_2(1.240)}, e_7(1.245)$	$e_4(1.235), e_5(1.235), e_6(1.235)$
	$e_3(\underline{1.240}, 1.735, 1.646)$	$e_1(1.235), e_3(1.485)$	$e_2(1.240), e_7(1.245)$	$e_4(1.235), e_5(1.235), e_6(1.235)$
	$e_4(1.485, 1.735, \underline{1.235})$	$e_1(1.235), e_3(1.485)$	$e_2(1.240), e_7(1.245)$	$e_4(1.235), e_5(1.235), e_6(1.235)$
	$e_5(1.485, 1.485, \underline{1.235})$	$e_1(1.235), e_3(1.485)$	$e_2(1.240), e_7(1.245)$	$e_4(1.235), e_5(1.235), e_6(1.235)$
	$e_6(1.735, 1.485, \underline{1.235})$	$e_1(1.235), e_3(1.485)$	$e_2(1.240), e_7(1.245)$	$e_4(1.235), e_5(1.235), e_6(1.235)$
	$e_7(1.735, \underline{1.240}, 1.646)$	$e_1(1.235), e_3(1.485)$	$e_2(1.240), e_7(1.245)$	$e_4(1.235), e_5(1.235), e_6(1.235)$
Round 2	$e_1(\underline{1.240}, 1.485, 1.980)$	$e_1(1.240), e_3(1.485)$	$e_2(1.240), e_7(1.245)$	$e_4(1.235), e_5(1.235), e_6(1.235)$
	...	...	...	...
	$e_7(1.735, \underline{1.240}, 1.646)$	$e_1(1.235), e_3(1.485)$	$e_2(1.240), e_7(1.245)$	$e_4(1.235), e_5(1.235), e_6(1.235)$

parameter β. Therefore, we introduce normalization parameter β to make the two parts become comparable, where β is the normalization parameter. When the preference coefficient $\alpha = 0.5$, the load balance part should equal to the replication factor part:

$$\begin{aligned} \beta \frac{1}{k} \sum_{i=1}^{k} l^2(p_i) &= \sum_{v \in V_b} |Rep(v)| \\ \Rightarrow \beta &= \frac{k \sum_{v \in V_b} |Rep(v)|}{\sum_{i=1}^{k} l^2(p_i)} \end{aligned} \tag{8}$$

Next we will give the valid variation range of β.

Proposition 3. *The maximum of $\sum_{i=1}^{k} l^2(p_i)$ is $|E_b|^2$, when all edges e in E_b is placed in the same partition. The minimum of $\sum_{i=1}^{k} l^2(p_i)$ is $\frac{|E_b|^2}{k}$, when all edges are evenly assigned to each partition.*

The proof of Proposition 3 can be found in Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2018.2890515.

Proposition 4. *The minimum of $\sum_{v \in V_b} |Rep(v)|$ is $|V_b|$, when all vertices are replicated exactly once. The maximum of $\sum_{v \in V_b} |Rep(v)|$ is $k|V_b|$, when all vertices are replicated by k times in the worst case.*

Proposition 5. *Base on Proposition 3, Proposition 4 and Eq. (8), we can conclude that $\beta_{max} = \frac{k^3|V_b|}{|E_b|^2}$, $\beta_{min} = \frac{k|V_b|}{|E_b|^2}$.*

4.3 Exact Potential Game

Now we are going to prove the edge partitioning game constructed above is an *Exact Potential Game*, which is guaranteed to have a *pure Nash Equalibrium* [16].

Definition 3. *Game $\mathcal{G} = \{\mathcal{N}, \mathbb{S}, \{C_i\}_{i \in \mathcal{N}}\}$ is an Exact Potential Game [16] if and only if there is a function $\Phi : \mathbb{S} \mapsto \mathbb{R}$ such that for $\forall i \in \mathcal{N}$,*

$$\Phi(S_i', S_{-i}) - \Phi(S_i, S_{-i}) = C_i(S_i', S_{-i}) - C_i(S_i, S_{-i}) \tag{9}$$

where $S_i' \neq S_i, S_i' \in \mathbf{S_i}$ is another strategy.

Theorem 1. *Game $EGP = \{E_b, \mathbb{S}, \{C_e\}$ is an Exact Potential Game.*

We postpone the proof for Theorem 1 in Appendix B, available in the online supplemental material.

Since each *Exact Potential Game* owns a pure Nash Equilibrium [16], we conclude that game $EPG = \{E_b, \mathbb{S}, \{C_e\}\}$ exists at least one pure Nash Equilibrium. Next we will evaluate the quality of these Nash Equilibriums.

4.4 Price of Anarchy

PoA (*Price of Anarchy*) is the worst ratio of social welfare $C(S)$ when the strategy profile is a Nash Equilibrium over the optimum value of $C(S)$. It is an important metric to measure the quality of Nash Equilibriums. For minimization cost social welfare objective, it is formally defined as:

Definition 4.

$$PoA = \frac{\max_{S \in PNE} C(S)}{OPT}, \tag{10}$$

where $PNE \subseteq \mathbb{S}$ is the set of pure Nash Equilibriums, OPT is the global minimum value of social welfare in the case of cost minimization problem.

Proposition 6. *Denote OPT_{min}^{BAL}, OPT_{min}^{REP} and OPT_{min} as the minimum value of $C^{BAL}(S)$, $C^{REP}(S)$ and $C(S)$ respectively, where $C^{BAL}(S) = \alpha\beta\frac{1}{k}\sum_{i=1}^{k} l^2(p_i)$, $C^{REP}(S) = (1-\alpha)\sum_{v \in V_b} |Rep(v)|$. Then we have:*

$$OPT_{min}^{BAL} + OPT_{min}^{REP} \leq OPT_{min} \tag{11}$$

Similarly, we can get $OPT_{max}^{BAL} + OPT_{max}^{REP} \geq OPT_{max}$ as well, where the OPT_{max}^{BAL}, OPT_{max}^{REP} and OPT_{max} are the maximum values of $C^{BAL}(S)$, $C^{REP}(S)$ and $C(S)$ respectively.

Proof. For the sake of brevity, we only prove Eq. (11).

Denote S^{*BAL}, S^{*REP} and S^* are the strategy profiles where the $C^{BAL}(S)$, $C^{REP}(S)$ and $C(S)$ get their minimum values respectively, then we have:

$$\begin{aligned} C^{BAL}(S^*) &\geq C^{BAL}(S^{*BAL}) \\ C^{REP}(S^*) &\geq C^{REP}(S^{*REP}) \end{aligned} \tag{12}$$

$$\begin{aligned} &\Rightarrow C(S^*) = C^{BAL}(S^*) + C^{REP}(S^*) \\ &\geq C^{BAL}(S^{*BAL}) + C^{REP}(S^{*REP}) \\ &\Rightarrow OPT_{min} \geq OPT_{min}^{BAL} + OPT_{min}^{REP}. \end{aligned} \tag{13}$$

□

Based on these results, we formulate the PoA of edge partitioning game as follows:

Proposition 7. *PoA of edge partitioning game $EPG = \{E_b, \mathbb{S}, \{C_e\}\}$ is bounded by k.*

The proof for Proposition 7 can be found in Appendix C, available in the online supplemental material.

5 Best Response Dynamics and Round Complexity

5.1 Best Response Dynamics Algorithm

In this section, we will show how to get a pure Nash Equilibrium based on *Best Response Dynamics* [17] and how fast it can converge into a Nash Equilibrium. The general *Best Response Dynamics* to find Nash Equalibrium is shown in Algorithm 1. To find a Nash Equilibrium for edge partitioning game, we propose a variant based on Algorithm 1. We show it in Algorithm 2.

In Algorithm 2, we use the partitioning results of Random strategy as the starting point of game process, where each edge randomly chooses a partition from the partition set with equal probability. For each $e \in E_b$, we compute the minimum cost and record the corresponding partition ID via τ (line 4-line 13). Note that in order to guarantee the *pure strategy* characteristic mentioned in Definition 1, when e gets the minimum in more than one partition, we break the tie with the minimum partition ID. If the partition with the minimum cost is not its current choice, then e changes its choice to get the minimum cost. The iteration subroutine (line 3-line 17) will continue proceeding until no edges change their partition choices in certain round.

Table 2 shows the computation of Algorithm 2 on Fig. 3. First, each edge randomly chooses an initial partition, for example, e_1 chooses partition p_1. Then in each round, every edge computes its best response based on the individual cost function (Eq. (6)), where it can get the minimum cost. The best response is marked with underline. For example, the best response for e_1 in Round 1 is p_1. The gray-filled cell pair in each row indicates an edge's (marked with bold font) strategy transformation. For example, e_2 changes its partition from p_1 to p_2 in Round 1. Note that no edge changes its current partition choice in Round 2, so a Nash Equilibrium is found in Round 2, i.e., the partitioning task of this batch is finished.

Algorithm 1. Best Response Dynamics

Input: Game $EPG = \{E_b, \mathbb{S}, \{C_e\}\}$, partition set $P = \{p_1, \ldots, p_k\}$.
Output: Nash Equilibrium.
1: Each edge e randomly chooses an initial partition from the partition set with equal probability.
2: **repeat**
3: **for** each edge $e \in E_b$ **do**
4: Find e's best response S';
5: **if** $S_e \neq S'$ **then**
6: $S_e \leftarrow S'$
7: **end if**
8: **end for**
9: **until** No edges change their current strategies.

5.2 Round Complexity

Let us analyze the round complexity of Algorithm 2. Denote the Harmonical Series $H(n) = \sum_{i=1}^{n} \frac{1}{i}$, then we define the potential function $\Phi(S)$ as:

$$\Phi(S) = \frac{1}{k}\alpha\beta \sum_{i=1}^{k} \sum_{j=1}^{l(p_i)} j + (1-\alpha) \sum_{v \in V_b} \sum_{i=1}^{k} H(d(p_i, v)). \quad (14)$$

In order to get the round complexity, we scale up the problem where the potential function $\Phi(S)$ takes integer values [2]. Specifically, we denote $\Phi_{\mathbb{Z}}(S) = c\Phi(S)$, where c is an integer such that $\Phi_{\mathbb{Z}}(S)$ only takes integer values.

Proposition 8. *The number of rounds required until game $EPG = \{E_b, \mathbb{S}, \{C_e\}\}$ converges to a pure Nash Equilibrium is $O(c|V_b| \ln |V_b| + c|V_b|)$.*

The proof for Proposition 8 is listed in Appendix D, available in the online supplemental material.

6 Experiments

In this section, we evaluate the performance of our partitioning strategy. Section 6.1 describes the experimental setup, including system model, environment description and datasets used. Section 6.2 describes how to preprocess the input in case it does not satisfy the assumption that the incident edges of a vertex are not present consecutively. Section 6.3 presents the time and memory consumption of the state-of-the-art partitioning methods based on the streaming, the quasi-streaming and the offline models, respectively. Section 6.4 compares the partitioning results with other five state-of-the-art streaming edge partitioning strategies based on the quasi-streaming model. Section 6.5 demonstrates the impact of batch size and the number of threads on our quasi-streaming model.

Algorithm 2. Edge Partitioning Game

Input: batch of edges E_b, partition set $P = \{p_1, \ldots, p_k\}$, perference factor α, normalization factor β.
Output: patitioning results for edges in this batch.
1: Each edge e randomly chooses an initial partition S_e from the partition set with equal probability.
2: **repeat**
3: **for** each edge $e = (u, v) \in E_b$ **do**
4: $minCost \leftarrow \infty$, $\tau \leftarrow 1$
5: **for** $i \in [1, k]$ **do**
6: compute $C_e(p_i, S_{-e})$ based on Eq. (6)
7: **if** $C_e(p_i, S_{-e}) < minCost$ **then**
8: $minCost \leftarrow C_e(p_i, S_{-e})$
9: $\tau \leftarrow i$
10: **else if** $C_e(p_i, S_{-e}) = minCost$ **and** $i < \tau$ **then**
11: $\tau \leftarrow i$
12: **end if**
13: **end for**
14: **if** $S_e \neq p_\tau$ **then**
15: $S_e \leftarrow p_\tau$
16: **end if**
17: **end for**
18: **until** No edges change their current partition choices.

6.1 Experiment Setup

We implemented a stand-alone version of graph partitioner that captures the behavior of distributed graph computing systems during the graph loading phase. All experiments (except the model comparisons in Section 6.2) are based on the quasi-streaming model illustrated in Fig. 2, which means all competitive strategies partition the whole graph in the batch based fashion. In other words, each batch is the basic unit of partitioning. When all batches finish their partitioning tasks, the whole graph's partitioning task is accomplished. For example, an edge adopting Greedy [21] strategy under the quasi-streaming model can only make use of the partition choices of edges inside the same batch. DBH [24] only uses degree information inside each batch. Other strategies carry on partitioning in a similar fashion. The data source and data sink in Fig. 2 correspond to the disk in our implementation.

All experiments were performed on a single machine, equipped with two fifteen-core Gold 6151 CPUs, extending to 60 vCPUs, 960 GB memory, running CentOs Linux Release 7.3 (kernel 3.10.0-514.el7) and JDK1.8.0. The maximum heap size of JVM is set to 950 GB. The real-world graph datasets [4], [13] and random graph datasets we used are listed in Tables 3 and 4 respectively. The random graph datasets are generated based on Erdös-Renyi random graph model via SNAP [13]. For all six partitioning strategies, in order to get rid of the impact from streaming order, we shuffle each batch before it is processed.

For the sake of convenience, we denote our game theory based strategy as Mint. Without explict declaration, the default parameters of our quasi-streaming model are set as follows. The batch size B is set to 6400 and the number of partitioning threads is set to 60, which is applied to all

TABLE 3
Description of Real-World Graphs

Alias	Graph	$\|V\|$	$\|E\|$	size
Arabic	arabic-2005 [4]	22 M	0.6 B	11 GB
UK	uk-2002 [4]	19 M	0.3 B	4.7 GB
Wiki	enwiki-2013 [4]	5.7 M	101 M	1.5 GB
LJ	soc-LiveJournal1 [13]	4.8 M	69 M	1.1 GB
Hollywood	hollywood-2009 [4]	1.1 M	113 M	1.5 GB
Weibo	rel_pre2	0.47 B	44.3 B	750.3 GB

Letter 'M' is the abbreviation of Million. Letter 'B' is the abbreviation of Billion and 'GB' represents GigaBytes.

TABLE 4
Description of Erdös-Renyi Random Graphs

$\|V\|$	$\|V\| = 50M$					
$\eta = \frac{\|E\|}{\|V\|}$	15	16	17	18	19	20
$\|E\|$	0.75 B	0.8 B	0.85 B	0.9 B	0.95 B	1 B
size	12.2 GB	13.1 GB	13.9 GB	14.7 GB	15.5 GB	16.3 GB

Larger η produces denser graphs.

competitive strategies. In addition, The default parameters of Mint strategy are set as $\alpha = 0.5$, $\beta = \frac{k^3|V_b|}{2|E_b|^2}$, where k is the number of partitions, $|E_b|$ and $|V_b|$ are the number of edges and vertices in each batch respectively.

We evaluate the performance of our approach by the metrics mentioned in Section 3.3.

6.2 Preprocessing

As mentioned in the introduction, our algorithms are based on the assumption that the edges incident on the same vertex are present consecutively in the input. As a result, if the input graphs do not meet this assumption, we need to have a preprocessing step.

The preprocessing scheme we used is a hash table that supports multi-threading. One vertex of an edge is used as the index key of the hash table, and the other vertex is added to a list corresponding to the index key. Thus the time complexity of the preprocessing step is $O(|E|)$ where $|E|$ means the total number of edges in the graph.

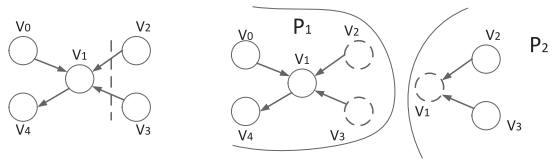
We designed an experiment to evaluate the actual time consumption of the preprocessing. First, we generated an Erdös-Renyi graph [13] where the incident edges of a vertex are stored next to each other; Second, we randomize all the edges in the graph; Third, the preprocessing step will be performed. The Erdös-Renyi graph we used and the time taken for preprocessing have been listed in Table 5. From this table, we can see that the execution time of the multi-threaded preprocessor is linearly related to the number of edges in the graph. In the following performance evaluations, we assume all graphs have been either preprocessed or satisfy the assumption that the edges incident on the same vertex are presented consecutively.

6.3 Performance Comparisons with Streaming and Offline Models

In this section, we will first compare our quasi-streaming model with the streaming and the offline models in terms of both time and memory consumption. Then we will compare the partitioning quality in terms of the replication factor and the standard deviation of the edge load between the quasi-streaming and the streaming models. The algorithm for the quasi-streaming model is Mint, and we pick the state-of-the-art algorithms HDRF [18] for the streaming model and METIS [11] for the offline model. As mentioned in Section 6.1, we use the default setting for Mint. For HDRF [18], we set the parameters $\lambda = 3$ and $\epsilon = 1$, which are the same as the authors' setting. For METIS, we use the default partitioning strategy stated on the manual.[1] The time and memory comparisons are illustrated in Figs. 5 and 6, respectively.

TABLE 5
Preprocessing Time on Various Erdös-Renyi Random Graphs

$\|V\|$	5 M	10 M	10 M	10 M	10 M	10 M
$\|E\|$	50 M	100 M	150 M	200 M	250 M	300 M
time	29 s	56 s	78 s	105 s	137 s	159 s



(a) Vertex partitioning via cutting edges. The cut edges are (v_2, v_1) and (v_3, v_1).

(b) The partitioning results of (a). The number of partitions is 2.

Fig. 4. Each edge cut will generate two replicated vertices in the worst case. However, vertices can share a replicated vertex in the same partition. For example, the replica of v_1 is shared by v_2 and v_3.

As Fig. 5 shows, the time taken by the streaming model is much longer than both the quasi-streaming and the offline models, and the gap is even increasing as the number partitions increases. In addition, the time consumed by the quasi-streaming model is comparable with the offline model.

As for the memory consumption, Fig. 6 shows that the memory occupied by both the streaming and the offline models are much larger than the quasi-streaming model, which corroborates the theoretical analyses mentioned in Section 1.

Note that although the quasi-streaming model excelled in both the time and memory consumptions on the streaming model, the partitioning quality is relatively worse than the streaming one which can be found in Tables 6 and 7. The reason is that, compared with the quasi-streaming model where each edge can only decide its partitioning choice within the corresponding batch consisting of a constant number of edges, in the streaming model, each edge can make use of all previous edges' partition choices. As a result, the streaming model can achieve nearly optimal edge load balance and smaller replication factors.

We did not compare the two partitioning metrics between the Mint algorithm in the quasi-streaming model and the METIS algorithm in the offline model. The reason is that, on one hand, the quasi-streaming model is close to streaming setting and far away from the offline model, since the batch size is a constant multiple of the number of partitions k. On the other hand, METIS is a vertex partitioning

1. http://glaros.dtc.umn.edu/gkhome/fetch/sw/metis/manual.pdf

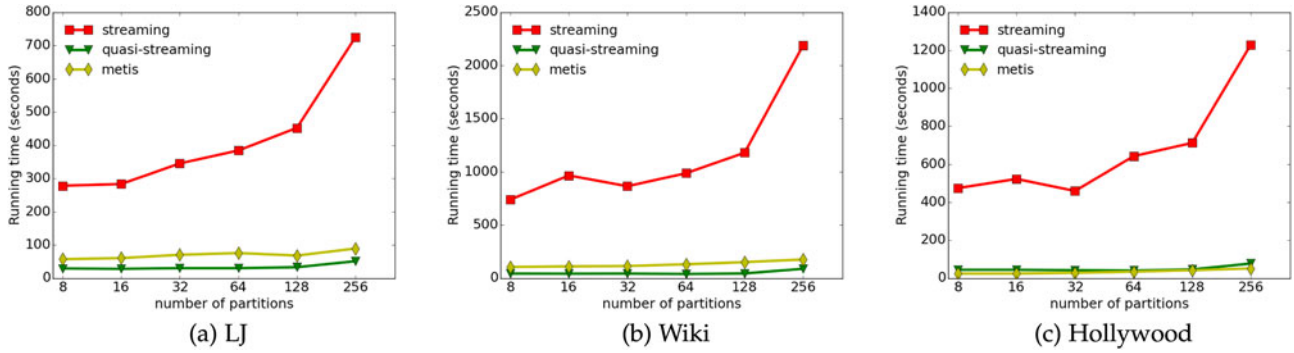


Fig. 5. Time comparison of quasi-streaming model (Mint), streaming model (HDRF), and offline model (METIS) on real-world graphs: LJ, Wiki and Hollywood.

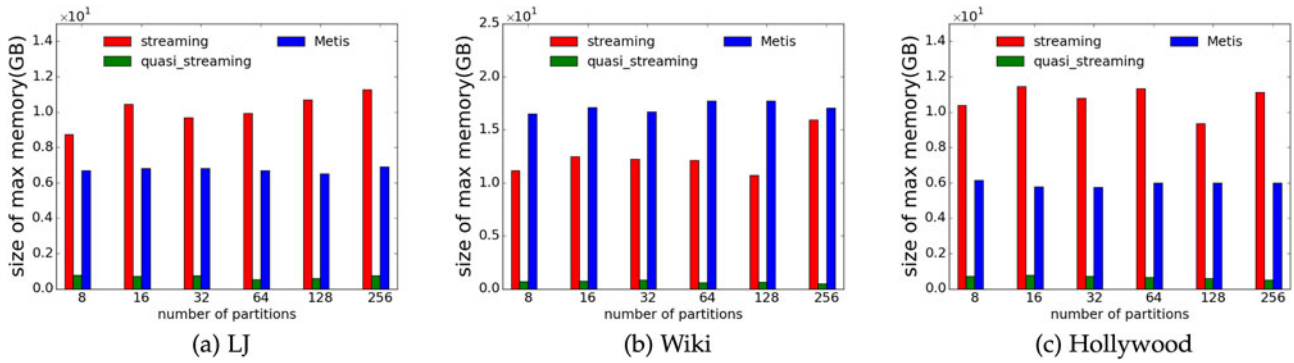


Fig. 6. Memory comparison of quasi-streaming model (Mint), streaming model (HDRF), and offline model (METIS) on real-world graphs: LJ, Wiki and Hollywood. The number of partitions varies in $[8, 256]$.

based strategy, where the partitioning quality is measured by edge cut. A straightforward observation is that each edge cut will generate two replication vertices in the worst case [18]. As a result, the replication factor of METIS can be roughly estimated as $\frac{2\times|E_{cut}|}{|V|}$, where E_{cut} is the set of edge cuts of METIS. However, this may be quite unfair for METIS, which is illustrated in Fig. 4. If we just assert each edge cut will generate two replicated vertices, we will get 4 replicated vertices in Fig. 4. But the actual number of replicated vertices is 3, since vertex v_2 and vertex v_3 can share the same replicated vertex of v_1 in partition p_2. Therefore, we don't compare the partitioning metrics with METIS.

TABLE 6
The σ Comparison between the Quasi-Streaming Model and the Streaming Model Where the Number of Partitions k is Fixed as 64

Graphs	WIKI	LJ	HOLLYWOOD
σ of quasi-streaming model	61.4	70.7	90.3
σ of streaming model	0.56	0.89	0.70

TABLE 7
The Replication Factor Comparison between the Quasi-Streaming Model and the Streaming Model Where the Number of Partitions k is Fixed as 64

Graphs	WIKI	LJ	HOLLYWOOD
Rep of quasi-streaming model	8.68	7.67	20.13
Rep of streaming model	5.09	3.04	6.58

6.4 Partitioning Quality Comparisons with Other Strategies within the Quasi-Streaming Model

In this section, we compare our solution's performance with five other state-of-the-art streaming partitioning strategies in terms of load balance metric σ and replication factor metric rep, which are defined in Section 3.3. The five other streaming partitioning strategies are Greedy [21], Fennel [23], DBH [24], HDRF [18] and Random strategy, where each edge chooses a partition randomly from the partition set with equal probability. Note that the first Greedy strategy for vertex partitioning was first proposed in [21]. Fennel is another vertex partitioning strategy. We refer to their edge partitioning counterparts provided in [5], [18]. The implementation details of the edge partitioning versions of Greedy and Fennel can be found in VGP[2] and BGEP,[3] respectively.

We use the Random strategy to get the intitial partitioning for Mint. For HDRF [18], we set the parameters $\lambda = 3$, $\epsilon = 1$ as same as the authors' setting. Considering the edge partitioning version of Greedy in [18] may give rise to severe load imbalance, we fine tune the original implementation via changing the balance factor λ from 1 to 3 for giving more attention to load balance part during the partitioning. DBH is a variant of hashing strategy, which leverages the skewed degree distribution in real-world graphs. Random strategy is similar to the hashing strategy, where each edge chooses a partition from partition set with equal probability. The parameters of system model and Mint

2. https://github.com/fabiopetroni/VGP
3. https://www.di.ens.fr/fbourse/publications/index.html

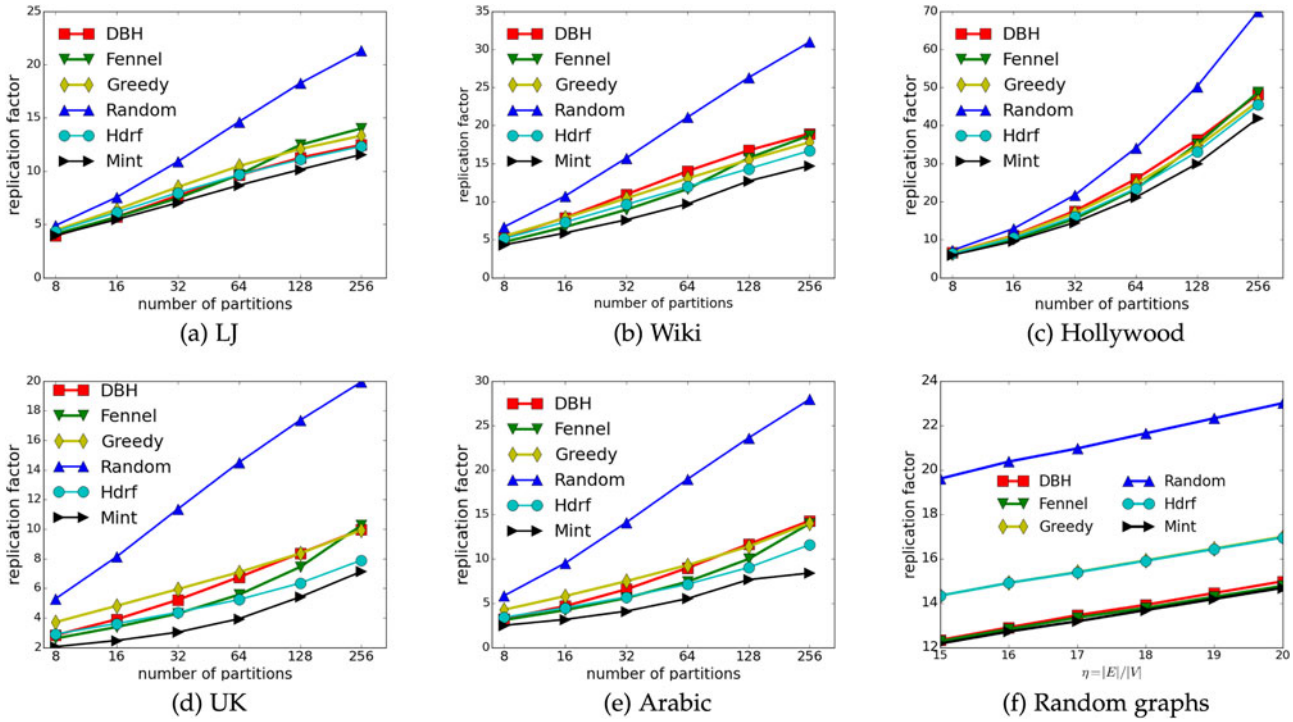


Fig. 7. Replication factor comparsions on real-world graphs and random graphs. The number of partitions varies in [8, 256]. As for random graphs' comparison, the number of partitions is fixed as 32.

follow the default settings in the experiment setup given in Section 6.1.

The replication factor comparisons of real world graphs (except for Weibo) and random graphs are illustrated in Fig. 7. As Fig. 7 shows, Mint can always achieve the smallest replication factor among all six partition strategies. To measure the improvement compared with the other five strategies, we define the replication factor improvement ratio as follows:

$$\gamma^{Rep}_{strategy} = \frac{Rep_{strategy} - Rep_{Mint}}{Rep_{strategy}} \times 100\%, \tag{15}$$

where $Rep_{strategy}$ denotes the replication factor of a specific strategy, $\gamma^{Rep}_{strategy}$ denotes the quantified improvement of Mint against the specific strategy in terms of replication factor. Then, the maximum and the average replication factor improvement ratio results through all datasets are listed in Table 8, where the number of partitions varies in $[8, 256]$. As Table 8 shows, Mint can achieve a reduction of replication factor up to 39.8 and 78.2 percent in terms of replication factor improvement ratio when compared with HDRF and Random strategy, respectively. Notice that replication factor is a harsh metric to minimize: when replication factor is decreased by one, the total number of replicated vertices will be reduced by $|V|$.

TABLE 8
The Replicatoin Factor Improvement Ratio of Mint Compared with the Other Five Partitioning Strategies

Strategies	DBH	Fennel	Greedy	Random	HDRF
max of $\gamma^{Rep}_{strategy}$	51.9%	38.5%	58.9%	78.2%	39.8%
mean of $\gamma^{Rep}_{strategy}$	22.9%	17.6%	28.2%	52.1%	19.3%

The σ metric comparison of real world graphs (except for Weibo) and random graphs are illustrated in Fig. 8. The σ metric of DBH is much larger than the other five strategies on Wiki graph, so we omit it from Fig. 8b. As Fig. 8 shows, Mint achieves far smaller σ metric compared with the other five strategies. Similarlly, to measure the improvement quantitatively, we define the σ metric improvement ratio as follows:

$$\gamma^{\sigma}_{strategy} = \frac{\sigma_{strategy}}{\sigma_{Mint}}, \tag{16}$$

where $\sigma_{strategy}$ denotes the load balance metric of a specific strategy, $\gamma^{\sigma}_{strategy}$ denotes the quantified improvement of Mint against the specific strategy in terms of load balance metric. The maximum and the average σ metric improvement ratio results are listed in Table 9, where the number of partitions varies in $[8, 256]$. As Table 9 shows, there is a case that Mint can achieve $830\times$ smaller σ metric than DBH, when comparing the two strategies on Wiki graph with 64 partitions. In this case, the σ metric of DBH and Mint are 50963.3199 and 61.3541, respectively. The significant reduction of σ metric experimentally demonstrates the tight relevance between social welfare function and QGEP's objective function, which is proved in Proposition 1.

Now we will show the partitioning performance on the Weibo graph under the quasi-streaming model. This graph consists of 0.47 Billion vertices and 44.3 Billion edges, which is around 500 times the Wiki graph in terms of graph size. According to Table 10, we can see that both the replication factor and the standard deviation of edge load of our Mint algorithm is better than those of the other five strategies. This is in line with Mint algorithm's partitioning results on the smaller size graphs shown before. In addition, the average time for processing the Weibo graph using different

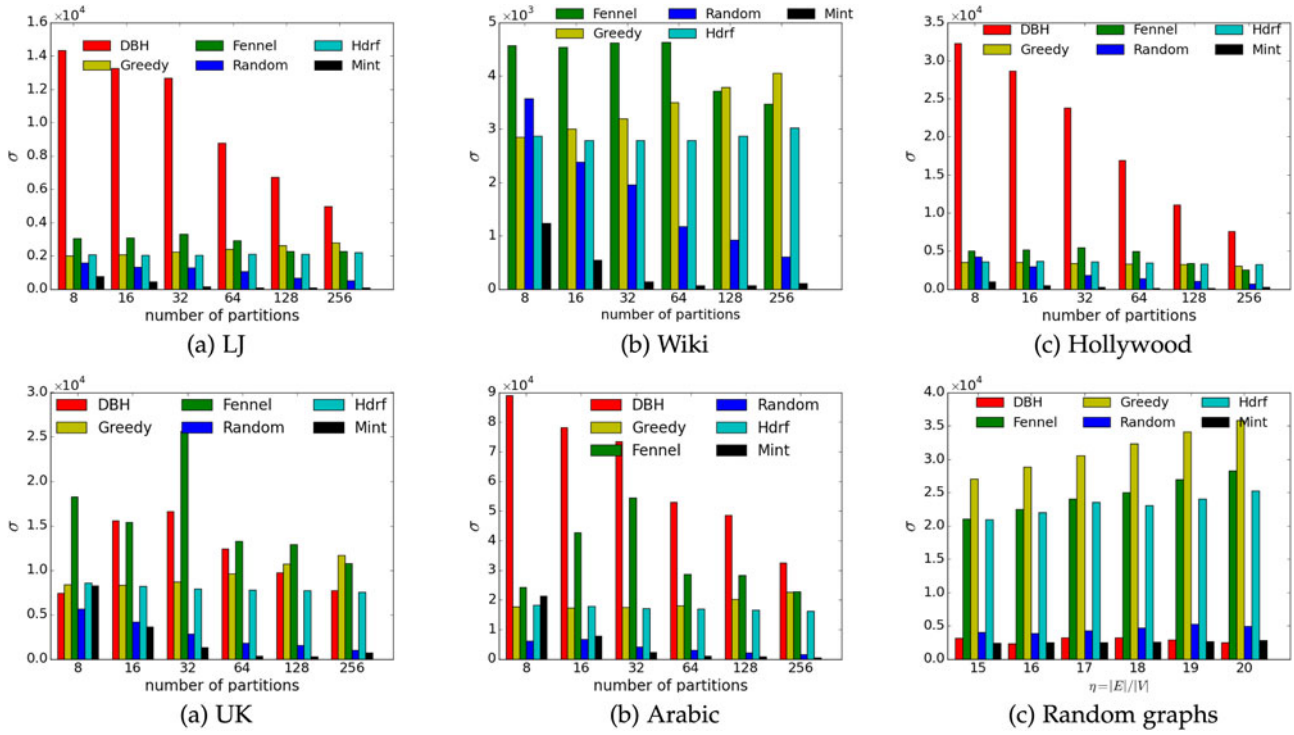


Fig. 8. σ (standard deviation of edge load) comparisons on real-world graphs and random graphs. Since the σ measure of DBH on Wiki graph is much larger than the others, we omit it from the comparison. The number of partitions varies in $[8, 256]$. As for random graphs' comparison, the number of partitions is fixed as 32.

strategies will take around 21 thousands seconds. This is also in accordance with the processing efficiency (the processing time over the number of edges) on smaller size graphs. For example, as will be shown in Fig. 11a, the average time for processing the Wiki graph takes around 47 seconds. Since the number of edges in Weibo graph is around 440 times of the Wiki graph, we will calculate the processing time of Weibo graph will be around 21 thousands seconds. This time consumption did not take into account the preprocessing since all the input graphs satisfy the assumption mentioned in the introduction and in Section 6.2. For huge graphs that do not satisfy the input assumption, according to Table 5, by simple calculations the preprocessing time on a huge graph with 44.3 Billion edges will be almost the same of the partitioning time on the graph.

6.5 The Impact of Batch Size and The Number of Threads on the Partitioning Quality within the Quasi-Streaming Model

Next we will show how the batch size B and the number of partitioning threads running concurrently affect the strategies' performance. The graphs used are LJ, Wiki, UK and Hollywood in the following experiments.

TABLE 9
The σ Metric Improvement Ratio of Mint Compared with the Other Five Partitioning Strategies

Strategies	DBH	Fennel	Greedy	Random	HDRF
max of $\gamma^{\sigma}_{strategy}$	830.7	75.5	61.5	19.1	46.6
mean of $\gamma^{\sigma}_{strategy}$	109.8	21.4	21.8	7.3	14.8

6.5.1 Impact of Batch Size

We now turn to evaluate the effect of batch size B. We fix the number of partitions to 32. The parameters' setting of all competitive strategies are the same with the configuration adopted in Section 6.4. The number of partitioning threads is 60, the same as the default setting. We only change the batch size B from 640 to 6400. For evaluating the effect comprehensively, we will illustrate how batch size B affects replication factor, σ of edge load and running time. The batch size B's impact on replication factor, edge load σ and running time are illustrated in Figs. 9, 10 and 11, respectively.

First of all, as Fig. 9 shows, the replication factors of all six strategies except the Random strategy decrease with the increasing of batch size. On the other hand, Fig. 10 shows that the σ metric of all six strategies decreases sharply along with the increasing of batch size except for the DBH and Random strategy. Finally, as Fig. 11 shows, all six strategies' running time almost stay invariant .

Now we analyze the reasons behind these results. For B's impact on replication factor, all five strategies can

TABLE 10
Partitioning Results on Weibo Graph When the Number of Partitions k is Fixed as 64

Strategies	DBH	Fennel	Greedy
rep	17.58	17.59	17.24
σ	232889.1	1230213.3	83865.5
$time(seconds)$	20809	21158	20987
Strategies	Random	HDRF	mint
rep	24.64	17.58	16.43
σ	210699.6	798400.7	21365.6
$time(seconds)$	21266	21628	21898

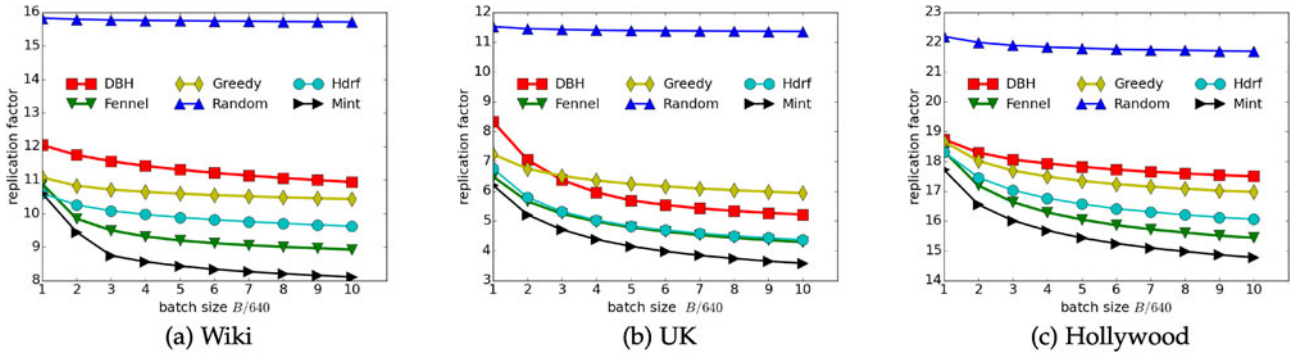


Fig. 9. Batch size B' impact on replication factor, where B varies from 640 to 6400. The number of partitions is 32.

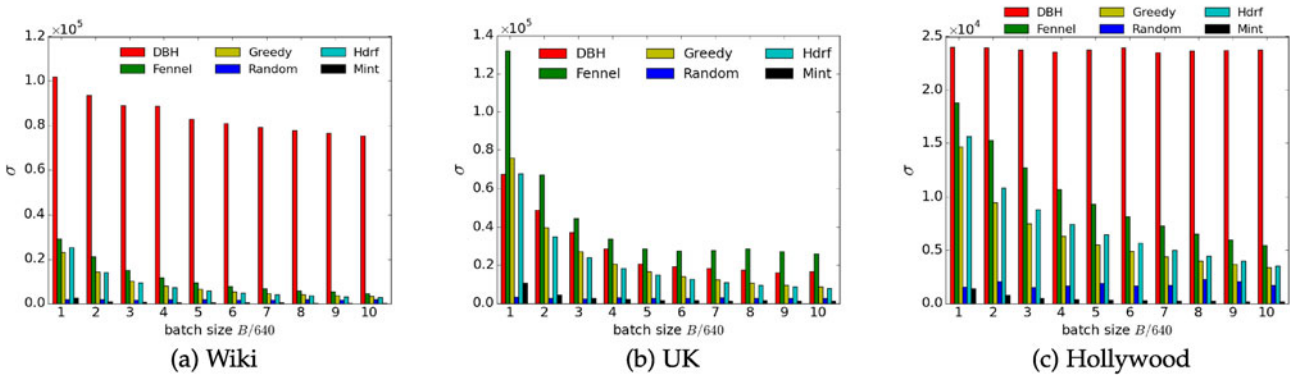


Fig. 10. Batch size B' impact on σ, where B varies from 640 to 6400. The number of partitions is 32.

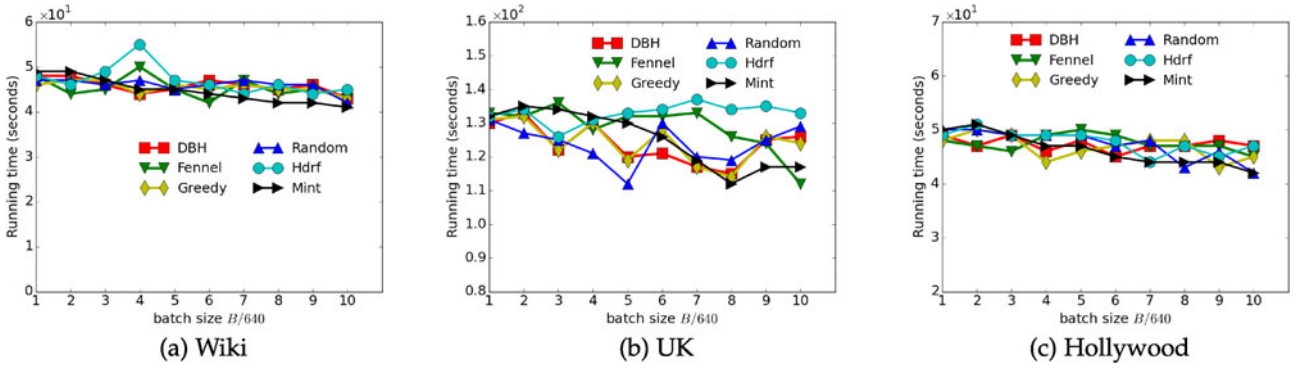


Fig. 11. Batch size B' impact on running time where B varies from 640 to 6400. The number of partitions is 32.

get more information except for the Random strategy when B increases. Much more information will help these strategies make better decisions. Note that Random strategy always chooses a partition with equal probability among all partitions no matter how big the batch size is. Therefore, batch size B almost has no effect on Random strategy in terms of both the replication factor metric and the σ metric. Since DBH chooses a partition for each edge by hashing the vertex ID with higher degree, bigger batch size almost has no effect on the σ metric of DBH. On the other hand, DBH is a degree based partitioning strategy, which could reduce the replication factor via cutting the high degree vertices as many as possible. Therefore, with a much bigger batch size, the paritial degree [18] information of vertices is more close to the real one, which can help DBH reduce the replication factor.

6.5.2 Impact of Number of Threads

To evaluate the number of partitioning threads' impact on Mint strategy, we adopt the default parameters' setting of Mint and only change the threads' number from 5 to 60 in the system model. In order to get the benefit of parallelism quantitatively, we define the speedup as the ratio of running time. The running time includes I/O time and partitioning time. The speedup for Mint algorithm on real world graphs is illustrated in Fig. 12. As the figure shows, the speedup is almost linear to the number of partitioning threads for the cases from partition number $k = 32$ to partition number $k = 64$. For the other cases, i.e., the partition number ranges from 4 to 16, the speedup does not increase linearly. For example, on Hollywood graph (Fig. 12c), Mint strategy can get its maximum speedup for partition number $k = 4$, $k = 8$, and $k = 16$ when the number of corresponding partitioning threads equals 15, 30, 40 respectively. After the

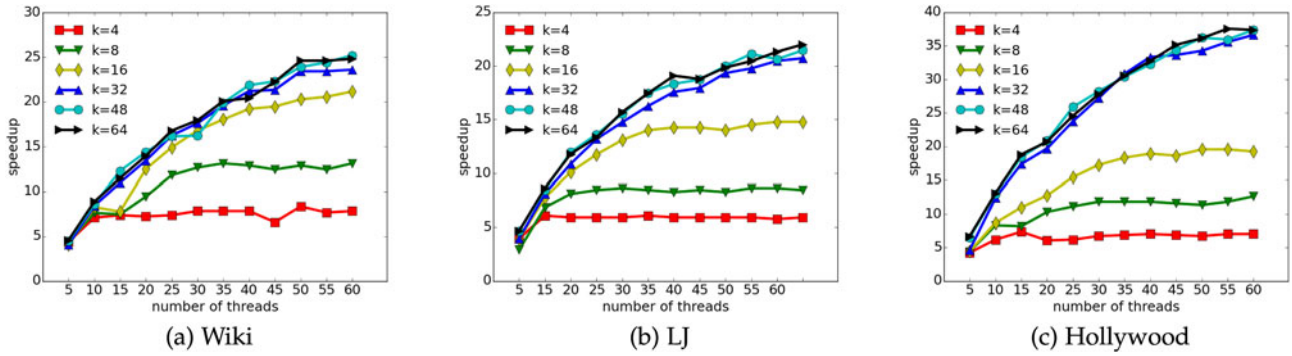


Fig. 12. Speedup for Mint algorithm on real world graphs: LJ, Wiki, Hollywood, where $\text{speedup of n threads} = \frac{\text{running time of n threads}}{\text{running time of one thread}}$. The number of partitions k varies in $[4, 64]$.

respective points, the speedup does not increase with more partitioning threads. Compared with the total number of threads, when the partition number is relatively small, the time cost of each round also weighs small (See Algorithm 2). As a result, the partitioning task of each batch can be finished quickly. When the Reader thread's feeding cannot keep up with the partitioning threads' processing in the system model (details refer to Fig. 2), race condition could occur frequently, which is a heavy burden to the underlying operating system. Consequently, the total running time could increase when severe race condition happens.

7 Conclusion

In this paper, we proposed the quasi-streaming model for edge partitioning problem, and a corresponding novel partitioning strategy based on game theory. In the quasi-streaming model, the edge stream is segmented into a series of batches where each batch size is a constant multiple of the number of partitions. Comparing with the streaming model, although each edge can have full knowledge of the edges within the same batch, each edge's decision cannot get benefits from the other edges' decisions outside of the batch. Thus performing efficient graph partitioning in the quasi-streaming model could be much harder than the streaming model and the offline model. We mathematically proved that the game process can always converge to a Nash Equilibrium. Then we measured the quality of these Nash-Equilibriums via PoA. The experiments show that our solution outperforms all other five exsiting streaming partitioning strategies in terms of load balance metric and replication factor metric on both real-world graphs and random graphs. Specifically, our game theory based solution can achieve 19.3 percent reduction on replication factor and 14.8× smaller load balance metric averagely when compared with the HDRF strategy, which is the state-of-the art streaming edge partitioning strategy. The significant reduction on replication factor and load balance metrics would improve the distributed graph computing systems' performance in terms of communication cost and computation balance. Finally, experiment results show that the game theory based strategy can get almost linear speedup related to the number of partitioning threads in the quasi-streaming model.

There are two directions in our future works. First, we intend to develop our solution based on ϵ-approximated Nash Equilibrium [17] instead of Nash Equilibrium, which may reduce convergence time sharply with modest loss on partitioning quality. Second, in order to evaluate how much speedup a specific application can get from our partitioning strategy, such as PageRank, we intend to apply our partitioning strategy to the real distributed graph computing systems [10] in the future work.

Acknowledgments

The first author thanks Liyuan Ouyang for helping to implement the experiment. This work is supported in part by National Key Research and Development Program of China under grant No. 2018YFB1003203, the National Natural Science Foundation of China Grants 61572216 and 61602195, and Fundamental Research Funds for the Central Universities, HUST: No. 2016YXMS075.

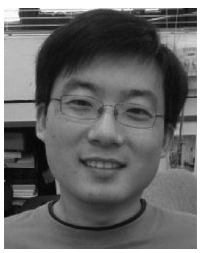
References

[1] K. Andreev and H. Räcke, "Balanced graph partitioning," in *Proc. 16th Annu. ACM Symp. Parallelism Algorithms Archit.*, 2004, pp. 120–124.
[2] N. Armenatzoglou, H. Pham, V. Ntranos, D. Papadias, and C. Shahabi, "Real-time multi-criteria social graph partitioning: A game theoretic approach," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2015, pp. 1617–1628.
[3] P. Boldi, B. Codenotti, M. Santini, and S. Vigna, "Ubicrawler: A scalable fully distributed web crawler," *Softw. Pract. Exper.*, vol. 34, no. 8, pp. 711–726, 2004.
[4] P. Boldi, and S. Vigna, "The WebGraph framework I: Compression techniques," in *Proc. 13th Int. World Wide Web Conf.*, 2004, pp. 595–601.
[5] F. Bourse, M. Lelarge, and M. Vojnovic, "Balanced graph edge partition," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2014, pp. 1456–1465.
[6] A. Buluç, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz, "Recent advances in graph partitioning," in *Algorithm Eng. - Select. Results Surveys*, 2016, pp. 117–158.
[7] R. Chen, J. Shi, Y. Chen, and H. Chen, "Powerlyra: Differentiated graph computation and partitioning on skewed graphs," in *Proc. 10th Eur. Conf. Comput. Syst.*, 2015, pp. 1:1–1:15.
[8] U. Feige, M. T. Hajiaghayi, and J. R. Lee, "Improved approximation algorithms for minimum-weight vertex separators," in *Proc. 37th Annu. ACM Symp. Theory Comput.*, 2005, pp. 563–572.
[9] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "Powergraph: Distributed graph-parallel computation on natural graphs," in *Proc. 10th USENIX Symp. Operating Syst. Des. Implementation*, 2012, pp. 17–30.
[10] H. Jin, P. Yao, and X. Liao, "Towards dataflow based graph processing," *SCIENCE CHINA Inf. Sci.*, vol. 60, no. 12, pp. 126102:1–126102:3, 2017.
[11] G. Karypis, and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 359–392, 1998.

[12] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *Bell Syst. Tech. J.*, vol. 49, no. 2, pp. 291–307, 1970.
[13] J. Leskovec, and R. Sosič, "Snap: A general-purpose network analysis and graph-mining library," *ACM Trans. Intell. Syst. Technol.*, vol. 8, no. 1, 2016, Art. no. 1.
[14] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, and J. M. Hellerstein, "Distributed graphlab: A framework for machine learning in the cloud," *Proc. VLDB Endowment*, vol. 5, no. 8, pp. 716–727, 2012.
[15] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: A system for large-scale graph processing," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2010, pp. 135–146.
[16] D. Monderer, and L. S. Shapley, "Potential games," *Games Economic Behavior*, vol. 14, no. 1, pp. 124–143, 1996.
[17] N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani, *Algorithmic Game Theory*. New York, NY, USA: Cambridge Univ. Press, 2007.
[18] F. Petroni, L. Querzoni, K. Daudjee, S. Kamali, and G. Iacoboni, "HDRF: Stream-based partitioning for power-law graphs," in *Proc. 24th ACM Int. Conf. Inf. Knowl. Manage.*, 2015, pp. 243–252.
[19] S. Sakr, F. M. Orakzai, I. Abdelaziz, and Z. Khayyat, *Large-Scale Graph Processing Using Apache Giraph*. New York, NY, USA: Springer, 2016.
[20] K. Schloegel, G. Karypis, and V. Kumar, "Parallel multilevel algorithms for multi-constraint graph partitioning (distinguished paper)," in *Proc. 6th Int. Euro-Par Conf. Parallel Process.*, 2000, pp. 296–310.
[21] I. Stanton, and G. Kliot, "Streaming graph partitioning for large distributed graphs," in *Proc. 18th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2012, pp. 1222–1230.
[22] A. Toshniwal, S. Taneja, A. Shukla, K. Ramasamy, J. M. Patel, S. Kulkarni, J. Jackson, K. Gade, M. Fu, J. Donham, N. Bhagat, S. Mittal, and D V. Ryaboy, "Storm@twitter," in *Proc. Int. Conf. Manage. Data*, 2014, pp. 147–156.
[23] C. E. Tsourakakis, C. Gkantsidis, B. Radunovic, and M. Vojnovic, "FENNEL: Streaming graph partitioning for massive scale graphs," in *Proc. of 7th ACM Int. Conf. Web Search Data Mining*, 2014, pp. 333–342.
[24] C. Xie, L. Yan, W.-J. Li, and Z. Zhang, "Distributed power-law graph computing: Theoretical and empirical analysis," in *Proc. Adv. Neural Inf. Process. Syst. 27*, 2014, pp. 1673–1681.
[25] X. Zhu, W. Chen, W. Zheng, and X. Ma, "Gemini: A computation-centric distributed graph processing system," in *Proc. 12th USENIX Symp. Operating Syst. Des. Implementation*, 2016, pp. 301–316.

Qiang-Sheng Hua received the BEng and MEng degrees from the School of Information Science and Engineering, Central South University, China, in 2001 and 2004, respectively, and the PhD degree from the Department of Computer Science, The University of Hong Kong, China, in 2009. He is currently an associate professor with Huazhong University of Science and Technology, China. He is interested in parallel and distributed computing, including algorithms and implementations in real systems. He is a member of the IEEE.

Yangyang Li received the BE degree from the School of Mathematics and Statistics, Huazhong University of Science and Technology (HUST), in 2015. He is currently working toward the master's degree in the Department of Computer Science, Huazhong University of Science and Technology (HUST). His research interests include streaming graph partitioning, social networks, and parallel computing.



Dongxiao Yu received the BSc degree from the School of Mathematics, Shandong University, in 2006 and the PhD degree from the Department of Computer Science, The University of Hong Kong, in 2014. He is currently a professor with the School of Computer Science and Technology, Shandong University. His research interests include wireless networks and distributed computing. He is a member of the IEEE.

Hai Jin received the PhD degree from Huazhong University of Science and Technology (HUST), in 1994. He is a professor with the School of Computer Science and Technology, Huazhong University of Science and Technology. He was postdoctoral fellow with the University of Southern California and The University of Hong Kong. His research interests include HPC, grid computing, cloud computing, and virtualization. He is an IEEE Fellow.