EasyChair Preprint

№ 1174

Local Graph Edge Partitioning with A Two-stage Heuristic Method

Shengwei Ji, Chenyang Bu, Lei Li and Xindong Wu

June 12, 2019

Local Graph Edge Partitioning with a Two-Stage Heuristic Method

Shengwei Ji [1,2], Chenyang Bu [1,2,3], *IEEE Member*, Lei Li [1,2,3], *IEEE Senior Member*, Xindong Wu [1,3,4], *IEEE Fellow*

[1] *Key Laboratory of Knowledge Engineering with Big Data (Hefei University of Technology),* Ministry of Education, China
[2] *School of Computer Science and Information Engineering, Hefei University of Technology,* Hefei, China
[3] *Institute of Big Knowledge Science, Hefei University of Technology,* Hefei, China
[4] *Mininglamp Academy of Sciences, Mininglamp Technology,* Beijing, China
Email: swji@mail.hfut.edu.cn, chenyangbu@hfut.edu.cn, lilei@hfut.edu.cn, xwu@hfut.edu.cn

***Abstract*—Graph edge partitioning divides the edges of an input graph into multiple balanced partitions of a given size to minimize the sum of vertices that are cut, which is critical to the performance of distributed graph computation platforms. Existing graph partitioning methods can be classified into two categories: offline graph partitioning and streaming graph partitioning. The first category requires global information for a graph during the partitioning, which is expensive in terms of time and memory for large-scale graphs. The second category, however, creates partitions solely based on the received edge information, which may result in lower performance than the offline methods. Therefore, in this study, the concept of local graph partitioning is introduced from local community detection to consider only local information, i.e., a part of the graph, instead of the graph as a whole, during the partitioning. The characteristic of storing only local information is important because real-world graphs are often large in scale, or they increase incrementally. Based on this idea, we propose a two-stage local partitioning algorithm, where the partitioning process is divided into two stages according to the structural changes of the current partition, and two different strategies are introduced to deal with the respective stages. Experimental results with real-world graphs demonstrate that the proposed algorithm outperforms the rival algorithms in most cases, including the state-of-the-art algorithm METIS.**

Keywords—Graph edge partitioning, Distributed graph computing, Local information

I. INTRODUCTION

With the advent of the big data era, graphs are used in a wide range of fields, such as social networks [1-3] and knowledge graphs [4, 5]. The scale of graph data has increased rapidly and has already exceeded the processing capabilities of single machines [6]. To achieve better performance, distributed graph computation systems that process large-scale graphs on a cluster of machines, such as Pregel [7], PowerGraph [8], GraphLab [9], and GraphX [10], have been proposed. For such systems, graph partitioning plays a significant role in improving their computing performance because it determines the computational workload of each machine and the communication between them.

In distributed graph computation systems, graph partitioning is classified into two types: vertex partitioning, and edge partitioning. Most of the traditional distributed graph computation systems, such as Pregel and GraphLab, use vertex partitioning, where vertices are evenly assigned to different partitions by cutting the edges. However, most real-world graphs follow the power-law distribution, i.e., most of the vertices have few relative neighbors, while a few vertices have many neighbors. In this case, vertex partitioning increases workload imbalance and communication overhead because of the high-degree vertices and the number of cross-partition edges [11]. Different from vertex partitioning, edge partitioning evenly assigns edges to different partitions by cutting vertices. Researchers have demonstrated that edge partitioning performs better than vertex partitioning on many real-world graphs [8, 10, 11]. Therefore, edge partitioning has been widely adopted in recent systems, including PowerGraph, GraphX, and Chaos [12].

One traditional approach, referred to as offline graph partitioning, is based on a global view of the graph. This method obtains high-quality partitions by using multiple iterations based on complete graph data and is widely used in distributed graph computation systems. As the scale of graph data has increased, the offline method has become unsuitable for large-scale graph partitioning because it is difficult to obtain global information for a graph [13]. Therefore, streaming graph partitioning has been proposed, which treats graph data as an online stream by reading the data serially, and then determining the target partition of a vertex when it is accessed [14]. With the streaming graph partitioning method, only partial graph data is needed, which is more suitable than the offline method for a large-scale graph. However, some shortcomings in the streaming heuristics have appeared. Firstly, compared with the offline method, the streaming method results in worse partitioning quality. Secondly, to provide maximum flexibility, the entire arrived graph data must be accessed [15], which means streaming graph partitioning also requires large portions of the graph.

To address the problems mentioned above, a local graph partitioning method is designed in this study. It relies only on the local information (i.e., a part of the graph) instead of the global information during the partitioning. Compared with the offline method, local graph partitioning is based on less graph information. Compared with the streaming method, local graph partitioning only needs to store data in memory for a single partition at most.

Based on local graph partitioning, a two-stage local partitioning (TLP) algorithm is designed. Most of existing graph partitioning algorithms adopt a single partitioning strategy while ignoring the influence of graph structure changes on the partitioning quality during the partitioning process. In this study, however, the concept of modularity is introduced from local community detection to quantify the structure of local partitions. We prove that the modularity of each partition is positively correlated with the partitioning quality. According to the structural changes of the local

partition, the partitioning process of each partition is divided into two stages, and different graph partitioning strategies are introduced at each stage. Experiments demonstrate that the proposed TLP algorithm performs well for graph data of different scales. The main contributions of this paper can be summarized as follows.

- A local graph partitioning method is designed to partition a graph using only local information. At the same time, the method needs to save data for only a single partition in memory, which is suitable for large-scale graph partitioning.
- To quantify the structure change of a graph during the partitioning process, the concept of modularity is introduced. Meanwhile, the modularity of each partition is proved to be positively correlated with the graph partitioning quality.
- Based on local graph partitioning, a new TLP algorithm is proposed. The partitioning process of each partition is divided into two stages according to modularity changes of local partitions. Each stage adopts one corresponding partitioning strategy.
- The TLP algorithm is tested on real-world graphs of different scales and is compared with several classic graph partitioning algorithms. The experiments demonstrate that TLP can achieve high-quality partitions on graphs of different scales.

The structure of this paper is as follows. Section II states the graph partitioning problem and introduces mainstream graph partitioning algorithms through two classification methods. The concept of modularity is also introduced. In Section III, a local graph partitioning method is designed, and a new TLP algorithm is proposed. In Section IV, the performance of the TLP algorithm is analyzed on an extensive set of real-world graphs and is compared with several state-of-the-art algorithms. Finally, Section V concludes the paper.

II. BACKGROUNDS

In this section, graph partitioning algorithms are first introduced through two different classifications. Then the concept of modularity is introduced.

A. Vertex Partition and Edge Partition

Graph data consists of vertices and edges; thus, graph partitioning can be classified into vertex partitioning and edge partitioning.

Definition 1. Cross-partition edge. The edge connecting two vertices that are allocated to different partitions in vertex partitioning.

Definition 2. Spanned vertex. The vertex is adjacent to two edges that are allocated to different partitions in edge partitioning.

Vertex partitioning refers to the allocation of all the vertices in a graph by cutting the edges. The objective of vertex partitioning is to minimize the number of cross-partition edges and balance the number of vertices between the partitions. To ensure that each partition can perform local calculations independently in a distributed graph computation system, each cross-partition edge generates a corresponding ghost (a local replica) [8]. The process of vertex partitioning is illustrated in Fig. 1(a). Edge $e_{a,b}$ and $e_{a,c}$ are the cross-partition edges, and the ghosts are the shaded vertices.

Edge partitioning refers to the even allocation of all the edges of a graph and allows vertices to span partitions. In edge partitioning, a spanned vertex generates a mirror (a local replica) of the vertex. The adjacent edges of vertex a in Fig. 1(b) are located in two partitions, resulting in a mirror shown as a shadow. The objective of edge partitioning is to minimize the number of mirrors and balance the number of edges between partitions.

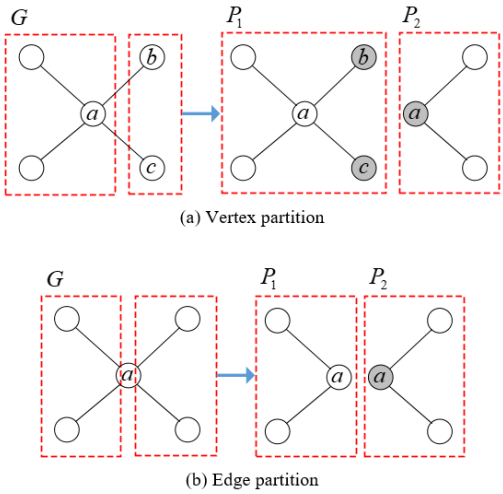

Fig. 1. Vertex vs. edge partitioning: (a) Graph G is partitioned into two partitions by cutting two edges, and the shaded vertices are ghosts; (b) Graph G is partitioned into two partitions by cutting one vertex, and the shaded vertex is a mirror.

In existing distributed graph processing systems, the computational load of machines is usually determined by the number of edges and the communication between the machines, which is related to the number of edges shared across machines. Therefore, the vertex partitioning algorithm usually leads to an uneven computing load, and a large number of edges shared across machines also blocks system communication [11]. Therefore, more and more distributed graph processing systems have begun to use edge partitioning to improve system efficiency.

Furthermore, the problem statement for graph edge partitioning is presented below.

Denote $G=(V, E)$ as an undirected graph with $n=|V|$ vertices and $m=|E|$ edges. For a subgraph S, denote $V(S)$ and $E(S)$ as the vertex set and edge set of S, respectively.

Definition 3. Balanced p-edge graph partitioning. Graph G is partitioned into p partitions. Each partition is denoted as P_k ($k \in \{1, 2, \ldots, p\}$). There are no duplicate edges between partitions, i.e., $E(P_i) \cap E(P_j) = \emptyset$ ($i, j \in \{1, 2, \ldots, p\}$, $i \neq j$), and $|E(P_k)| \leq C$, where C is the maximum capacity for edges in each partition.

Definition 4. Replication factor (*RF*) [13, 16]. To quantify the number of mirror vertices, the *RF* is defined as

$$RF = \frac{\sum_{k=1}^{p} |V(P_k)|}{|V|} \quad (1)$$

The graph edge partitioning problem consists of two aspects, such that 1) each partition load is within a given bound, and 2) the number of mirror vertices is minimized. Hence, graph edge partitioning can be defined with Definition 5. In Table I, we give an overview of the notation used in this paper, in order of presentation.

Definition 5. Graph edge partitioning. The graph edge partitioning problem seeks to find a balanced p-edge graph partitioning to minimize the RF.

TABLE I. NOTATION OVERVIEW

$G=(V, E)$	Graph with a set of vertices V and edges E.
P_k	The kth partition of G.
$e_{a,b}$	An edge connecting vertices a and b.
$n=\|V\|$	The number of vertices in V.
$m=\|E\|$	The number of edges in E.
$V(S)$	The vertices set of a subgraph S.
$E(S)$	The edges set of a subgraph S.
p	The number of partitions.
C	The maximum capacity for edges in each partition.
RF	The replication factor of a graph edge partitioning.
$M(P_k)$	The modularity of P_k.
$E_{out}(P_k)$	The set of external edges in P_k.
$N(v_i)$	The set of vertices that are adjacent to vertex v_i.
$N(P_k)$	The set of vertices that are adjacent to any vertices in P_k.
$\mu_{s1}(v_i)$	The criterion for selecting vertex v in Stage I.
$\mu_{s2}(v_i)$	The criterion for selecting vertex v in Stage II.
d	The average degree of the vertices in G.
L	The maximum number of vertices in each partition.
R	The ratio parameter of two stages.

B. *Offline Partition and Streaming Partition*

From another classification perspective, graph partitioning algorithms are classified into offline graph partitioning algorithms and streaming graph partitioning algorithms.

Offline graph partitioning algorithms are based on global information and are usually adopted in early distributed graph computation systems. As shown in Fig. 2(a), offline graph partitioning requires complete graph data before the partitioning process. For example, the classic algorithm Kernighan-Lin (KL) [17] partitions the graph into two parts initially, and exchanges arbitrary pairs of vertices between the two parts to find the optimal solution. Based on the global view of the graph, the KL algorithm can obtain a good result if there is good initialization. METIS [18] adopts a multi-level partitioning scheme, which includes the following three steps: coarsening to reduce the size of the graph; partitioning the reduced graph; decoarsening to map partitions back to the original graph. This leads to state-of-the-art quality partitions on a great number of graphs [19].

Different from offline graph partitioning, streaming graph partitioning assumes that the graph data arrives in a stream, and the target partition is determined as the data arrives. As shown in Fig. 2(b), the edges arrive in the order of $e_1, e_2, e_3, \ldots$. When an edge arrives, it can be allocated to the target partition. For example, $e_1$is allocated to P_1, and e_2 is allocated to P_2, etc.

The classic streaming graph partitioning algorithms such as LDG [15] and FENNEL [20] have greedy policies using different heuristics to deal with the received graph data. However, their precision is lower than that of METIS. Almost all streaming graph partitioning algorithms have the following characteristics: 1) When the data arrives, it is immediately allocated to the target partition, and it is not moved after it is placed. 2) Only the received data is accessed for partitioning.

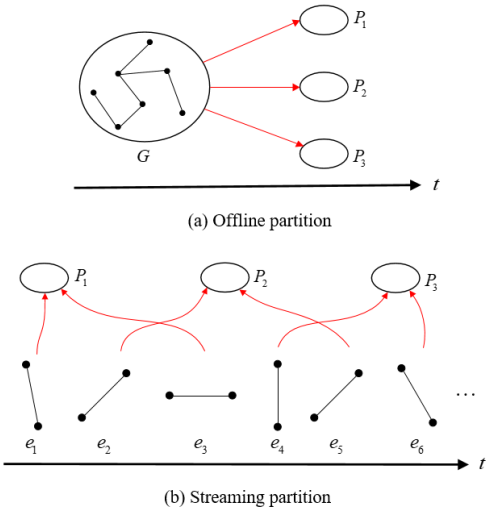

Fig. 2. Processes of offline graph partitioning and streaming graph partitioning: (a) After the graph G is obtained, it is partitioned into three partitions; (b) The graph data arrives in an edge stream, and the target partition is determined after each edge arrives.

In general, offline graph partitioning algorithms with high partitioning accuracy cannot deal with large-scale graphs because they require complete graph information. Instead, large graphs can be partitioned quickly using streaming graph partitioning algorithms based on partial graph information. However, the performance of streaming partitioning is poor, and all the received data must be saved. As the amount of received data gradually increases, the data that must be saved gradually increases, which means large portions of the graph are required.

C. *Modularity*

The concept of modularity was commonly used in the community detection area. For example, Luo et al. introduced the concept of modularity M to measure the quality of the detected communities for local community detection [21]. The greater the value of M, the better the detected communities. Luo et al. use modularity to analyze the characteristic of the local community in the process of community detection [22]. Jie et al. use weighted modularity to find crisp and fuzzy communities in undirected and unweighted networks [23]. In this study, we introduce the concept of modularity to graph partitioning problems. Related concepts are defined as follows.

Definition 6. Internal edge. The edge in one partition connecting two vertices that both belong to this partition.

Definition 7. External edge. The edge in one partition connecting two vertices where one of the vertices is in the same partition, and one is not.

Definition 8. Modularity [22]. The modularity of one partition is the ratio of the internal edges to the external edges, which is denoted as Eq. (2).

$$M(P_k) = \frac{|E(P_k)|}{|E_{out}(P_k)|} \quad (2)$$

where $E(P_k)$ is denoted as the set of internal edges in P_k. $E_{out}(P_k)$ is denoted as the set of external edges in P_k. $|E(P_k)|$and $|E_{out}(P_k)|$ represents the number of edges in $E(P_k)$ and $E_{out}(P_k)$, respectively. With the graph partitioning, the number of internal edges and external edges in one partition is continually changing. When the $M(P_k)$ is small, the partition structure is loose. As the number of edges allocated to P_k gradually increases and the value of $M(P_k)$ gets larger, P_k becomes tighter.

Furthermore, to affirm the relationship between modularity and the structure of each partition, we prove the modularity of each partition is positively correlated with the graph partitioning quality in section III.

III. Proposed Algorithms

In this section, a TLP algorithm is proposed. Section *A* introduces the motivation for the algorithm. In Section *B*, the frame of the TLP algorithm is presented. Sections *C* and *D* illustrate respective partitioning strategies for the two stages. Section *E* presents the complexity analysis of the TLP algorithm.

A. Motivation

Local Graph Partitioning. As mentioned in Section 2.3, the current graph partitioning algorithms can be classified into offline graph partitioning algorithms and streaming graph partitioning algorithms. The former algorithms have higher partitioning precision, but they need complete graph data before partitioning begins. The latter algorithms can partition graphs according to partial graph data, but they need to save all the received data, and the partitioning quality is worse than with offline heuristics.

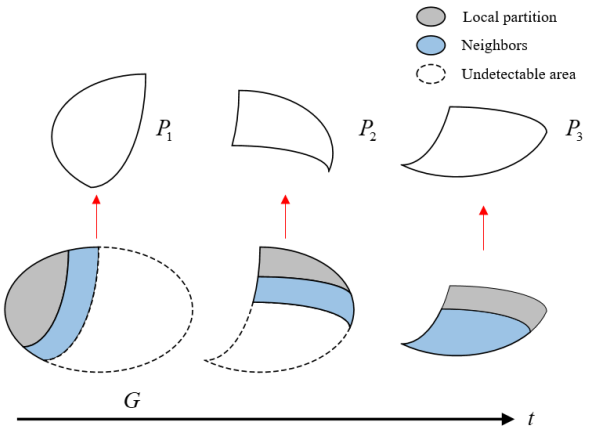

Fig. 3. Processes of local graph partitioning. Graph G is partitioned into several parts sequentially. In each partition, a single vertex constitutes the initial local partition at first. Then, one vertex is added into the local partition at each step until the local partition is full. Only the current local partition and neighbors are accessed in each round.

Considering the limits of the above two methods, a local graph partitioning method is proposed in this paper. As shown in Fig. 3, a graph is partitioned into three partitions sequentially. In round 1, the initially empty local partition is expanded in steps when $|E(P_1)| \leq C$. In each step, one optimal vertex is selected from the neighbor vertex set of P_k by a heuristic method, and then the edges between the optimal vertex and P_1 are allocated to P_1. After round 1 is over, round 2 will start with a new vertex to obtain partition P_2. The graph partitioning is complete when all three rounds have been completed.

Through the above analysis, the characteristics of the local graph partitioning method can be summarized as follows.

- Local graph partitioning relies only on information about the local partition, which is suitable for large graphs.
- Only one partition is obtained per round. P_k will not change once round k is over, which means that only data for one partition needs to be saved in memory. Therefore, this method can effectively reduce stored data.

Two stages. Most of existing graph partitioning algorithms adopt a single partitioning strategy during the whole partitioning process. However, the change in graph structure caused by each partition can influence the partitioning quality. For example, in the initial partitioning process, the graph structure of a partition can be loose because the number of edges is relatively small. With the graph partitioning, the partition can gradually be more compact.

To quantify the structure of the partition, we introduce the concept of modularity from the field of local community detection [21, 22]. Modularity was originally proposed to detect the local community. In this study, however, modularity is used to measure the structure of each partition. To improve the accuracy of local graph partitioning, the partitioning process is divided into two stages according to its modularity. In Stage I, a graph partitioning strategy is proposed to choose the closest and the local maximal-degree vertex. In Stage II, the vertex that makes the local partition tightest is selected based on the other partitioning strategy.

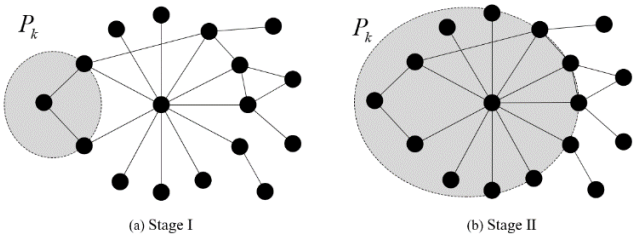

Fig. 4. The local partition in two stages: (a) The modularity of the local partition P_k is small in the initial stage, and the vertexes with larger degrees are selected as the core vertexes of the partition P_k; (b) The structure of P_k is tight in stage II, and the vertexes close to P_k are chosen.

As shown in Fig. 4(a), the modularity of the local partition P_k is small in the initial stage. In this case, the core vertexes with the large degrees in the graph are regarded as more proper to be selected. In Figure 4(b), as the local partition structure in stage II is tighter than in stage I, the vertexes close to P_k are chosen. The quality of graph partitioning can be effectively improved with this two-stage graph partitioning method, as shown by the experiments.

B. TLP Algorithm

In this subsection, a two-stage local partitioning (TLP) algorithm is proposed. In order to quantify the structure

change of each partition with the modularity, we first prove the correlation between the modularity of each partition and the quality of graph partitioning, which is given as follows.

Claim 1. For any graph G and any positive integer p, the modularity of P_k is negatively correlated with the replication factor of the balanced p-edge graph partitioning.

Proof. An averaging argument is used. Given a graph $G=(V, E)$ with average degree d, the number of partitions is p. We have

$$|V| \times d = \mathbf{2}|E|. \tag{3}$$

Assume that the number of edges in each partition is equal in balanced p-edge graph partitioning. Then,

$$|V| \times d = \mathbf{2}p \times |E(P_k)|. \tag{4}$$

As for P_k, we also have

$$|V(P_k)| \times d = \mathbf{2}(|E(P_k)| + |E_{out}(P_k)|) \ . \tag{5}$$

Combining Eq. (4) and Eq. (5), we obtain

$$\begin{aligned} &\frac{\sum_{k=1}^{p}|V(P_k)|}{|V|} \\ &= \frac{\sum_{k=1}^{p}(|E(P_k)| + |E_{out}(P_k)|)}{|E|} \ . \\ &= 1 + \frac{1}{p} \times \sum_{k=1}^{p} \frac{1}{M(P_k)} \end{aligned} \tag{6}$$

From Eq. (6), we can deduce that the larger the modularity of each partition, the smaller the *RF*, which completes the proof.

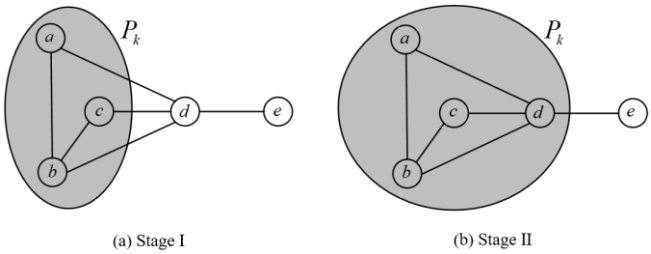

Fig. 5. Two stages of local graph edge partitioning: (a) In stage I, the modularity of P_k is smaller than 1, which means the partition structure is loose; (b) In stage II, the modularity of P_k is not less than 1, which means the partition structure is compact.

At the initial partitioning of P_k, the number of allocated edges in P_k is small, and the number of external edges in P_k is large. At this time, the partition structure is loose, thus $M(P_k)$ is small. As the number of edges allocated to P_k gradually increases and the value of $M(P_k)$ gets larger, P_k becomes tighter. In the TLP algorithm, the partitioning process of P_k is divided into two stages according to $M(P_k)$. The criteria for the two-stage division is shown in Table II.

TABLE II. DIVISION INTO TWO STAGES BASED ON MODULARITY

Stage	Criteria
Stage I	$0<M(P_k)\leq 1$
Stage II	$1\leq M(P_k)$

When $0<M(P_k)\leq 1$, the process of graph partitioning is in Stage I where there are more internal edges than external edges in P_k. The process of graph partitioning is in Stage II when $M(P_k)\geq 1$, where the partition becomes tighter. Figure 5 displays an example of both stages. In Fig. 5(a), $|E(P_k)|=2$, $|E_{out}(P_k)|=3$, and $M(P_k)=0.67$, so the partitioning is in Stage I. In Fig. 5(b), $M(P_k)=5$, thus the partitioning is in Stage II.

Different graph partitioning strategies for Stage I and Stage II are proposed in subsection *C* and *D*, respectively. The TLP algorithm for one partition is shown in Algorithm 1.

Algorithm 1 TLP for one partition

```
1. Select vertex x from G randomly;
2. P_k ← ∅;
3. N(P_k) ← N(x);
4. while |E(P_k)| ≤ C do
5.    if M(P_k) ≤ 1 do  //Stage I
6.       Choose vertex v from N(P_k) according to Section 3.3;
7.    else do          //Stage II
8.       Choose vertex v from N(P_k) according to Section 3.4;
9.    end if
10.   Allocate edges between v and vertices in P_k;
11.   if N(P_k) is empty do
12.      break
13.   end if
14. end
```

C. Graph Partitioning Strategy in Stage I

The graph partitioning strategy in Stage I selects the optimal vertex v from $N(P_k)$ that is close to P_k and has a high degree. The definition of $N(P_k)$ is as follows.

The criterion $\mu_{s1}(v_i)$ [22] for selecting vertex v from $N(P_k)$ is as shown in Eq. (3).

$$\mu_{s1}(v_i) = \max_{v_j \in N(v_i) \cap P_k} \frac{|N(v_i) \cap N(v_j)|}{|N(v_j)|} \tag{7}$$

where $|N(P_k)|$ is the number of neighbor vertices of v_i. The closeness between v_i and partition P_k is measured by the closeness between v_i and the vertices in partition P_k according to Eq. (1). At the same time, when the degree of v_i is large, the greater the number of neighbors of v_i, the larger the value $|N(v_i) \cap N(v_j)|$ may be. By calculating $\mu_{s1}(v_i)$ of each vertex in neighbor vertex set $N(P_k)$, the selection strategy of the optimal vertex v is defined as follows.

$$v = \arg\max_{v_i \in N(P_k)} \mu_{s1}(v_i) \tag{8}$$

That is, the vertex with the largest value of $\mu_{s1}(v_i)$ in neighbor vertex set $N(P_k)$ is selected as the optimal vertex v. Then, the edges between v and the vertices in partition P_k are allocated to P_k. To give an example, the graph partitioning strategy in Stage I is explained. From Fig. 6(a), it can be observed that vertices a, e, and g are in the neighbor vertex set $N(P_k)$. According to Eq. (3), we can calculate that $\mu_{s1}(a)=0.4$, $\mu_{s1}(b)=0.6$, and $\mu_{s1}(g)=0.5$.

Then, vertex e is selected as the optimal vertex according to Eq. (4). As can be seen, the degree of vertex e and g are the same, while there are more edges between e and the vertices in P_k than between g and the vertices in P_k. The number of edges between e and the vertices in P_k is equal to that between a and the vertices in P_k, while the degree of e is higher than

that of a. Therefore, the graph partitioning strategy in Stage I achieves the selection of a vertex close to P_k with a high degree.

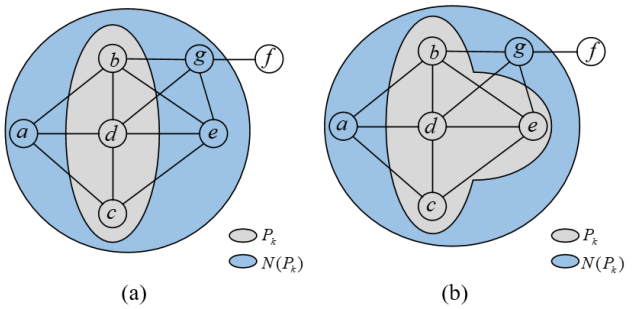

Fig. 6. Graph partitioning strategy in Stage I: (a) There exist vertices *a, g, e* in the neighbor vertex set $N(P_k)$. The optimal vertex is selected from the neighbor vertex set $N(P_k)$ based on the value of $\mu_{s1}(v_i)$ which is calculated by Eq. (7); (b) Allocate the edges between vertex *e* and partition P_k, because the vertex e is selected.

D. Graph Partitioning Strategy in Stage II

In Stage II, the local partition becomes tighter with the expansion by adding the optimal vertex v from $N(P_k)$. The selection criterion $\mu_{s2}(v_i)$ is based on the change in modularity. The representation of $\mu_{s2}(v_i)$ is:

$$\mu_{s2}(v_i) = 1 - \frac{1}{1+\Delta M}, \tag{9}$$

where ΔM is defined as in Eq. (6).

$$\Delta M = M'(P_k) - M(P_k), \tag{10}$$

where $M(P_k)$ is the modularity before selection is performed. $M'(P_k)$ is the modularity of the partition if there is a vertex v_i allocated to P_k. The partition will be tighter if the optimal vertex is added with the largest value of $\mu_{s2}(v_i)$. Thus, the selection strategy of the optimal vertex v in Stage II is defined as follows.

$$v = \arg\max_{v_i \in N(P_k)} \mu_{s2}(v_i) \tag{11}$$

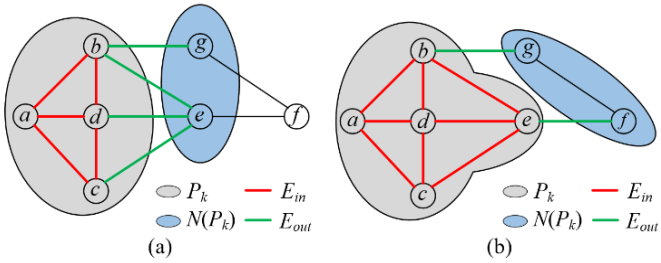

Fig. 7. Graph partitioning strategy in Stage II: (a) There exist vertices *g, e* in the neighbor vertex set $N(P_k)$. The optimal vertex is selected from the neighbor vertex set $N(P_k)$ based on the value of $\mu_{s2}(v_i)$ which is calculated by Eq. (9); (b) Allocate the edges between vertex *e* and partition P_k, because the vertex *e* is selected.

The edges between v and the vertices in P_k are then allocated to P_k. The graph partitioning strategy in Stage II will be explained through an example shown in Fig. 7. Before the allocation, $|E_{in}|$=5, $|E_{out}|$=4, and the modularity of P_k is $M(P_k)$=1.25. At this time, $N(P_k)$={*g, e*}. Assume that vertex *g* is added to P_k, $|E'_{in}|$=6, $|E'_{out}|$=4, $M'(P_k)$=1.5, and $\Delta M(g)=M'(P_k)-M(P_k)$=0.25; vertex *e* is added to P_k, $|E''_{in}|$=8, $|E''_{out}|$=2, $M''(P_k)$=4, and $\Delta M(e)=M''(P_k)-M(P_k)$=2.75. Vertex *e* can make P_k tighter because $\mu_{s2}(e) > \mu_{s2}(g)$. Therefore, *e* is the optimal vertex in $N(P_k)$. The edges connecting *e* and the vertices in P_k are then allocated to P_k. The partition result after allocation is shown in Fig. 7(b).

E. Analysis of TLP

Two characteristics of the TLP proposed are summarized as follows.

- The graph partitioning process relies only on local information using a local graph partitioning method.
- Only one partition must be saved in memory.

However, there also exist some limitations in the proposed algorithm.

- The graph must be traversed in BFS (Breadth First Search) order when the partition P_k expands.
- The selection of the optimal vertex in $N(P_k)$ requires traversing all the vertices in $N(P_k)$, which may degrade time performance when $N(P_k)$ is very large.

Time and space complexity. Denote d as the average degree of the vertices in the graph G, and L as the maximum number of vertices in each partition. At first, there is only one vertex in partition P_k, and the algorithm traverses d vertices to determine which vertex to select into P_k. There are now two vertices in P_k, and the algorithm should traverse at most $2d$ vertices, and so on. In general, when there are L vertices in P_k, Ld vertices at most should be traversed. Furthermore, we need $O(d)$ for each vertex computation in $N(P_k)$. Because $\sum_{i=1}^{L} i * d * d = d^2 L(L+1)/2$, the time complexity of our algorithm is $O(L^2 d^2)$. The space complexity is $O(Ld)$ because we need to store only P_k and $N(P_k)$.

For the state-of-the-art graph partitioning algorithm METIS [18], the time complexity is $O(n+m+k\log(k))$, and the space complexity is $O(n)$. Although the time complexity of TLP is slightly higher than that of METIS, the space complexity of TLP is much lower than that of METIS. In Section IV, the partitioning quality of two algorithms will also be compared.

IV. Experiments

In this section, the performance of our algorithm, analyzed through experiments, is discussed. First, the evaluation value, experimental platform, and real-world datasets are introduced. In Section 4.2, the proposed TLP algorithm is compared with four other algorithms. In Section 4.3, the two-stage method is compared with the one-stage method by redefining the division criterion of the two stages.

A. Setup

Evaluation. The *RF* [13] is illustrated in Eq. (1) as a measurement of the quality of graph partitioning. The greater the number of spanned vertices in each partition, the larger the *RF* will be. The minimum *RF* is *RF*=1, which means there is no spanned vertex in any partition.

Experimental Environment. The TLP algorithm proposed in this paper was implemented in Python. We evaluate all graph partitioning algorithms on a machine with an Intel i7-8700k 3.70 GHz Core processor and 48 GB RAM.

Datasets. We used nine real-world graph datasets for our experiments. The statistics for the graphs are listed in Table

III. Graphs G_1-G_8 can be found in SNAP [24], and G_9 comes from the huapu system [25].

TABLE III. REAL-WORLD GRAPH DATASETS

Graph Name	Notations	$\|V(G)\|$	$\|E(G)\|$	$\|V(G)\|+\|E(G)\|$
email-Eu-core	G_1	1,005	25,571	26,576
Wiki-Vote	G_2	7,115	103,689	110,804
CA-HepPh	G_3	12,008	118,521	130,529
Email-Enron	G_4	36,692	183,831	220,523
Slashdot081106	G_5	77,357	516,575	593,932
soc_Epinions1	G_6	75,879	508,837	584,716
Slashdot090221	G_7	82,144	549,202	631,346
Slashdot0811	G_8	77,36	905,468	905,468
huapu	G_9	4,309,321	7,030,787	11,340,108

B. Performance Comparison

In this subsection, we discuss the testing of the TLP algorithm on different graphs and compare it with several state-of-the-art graph partitioning algorithms. The comparison algorithms used in this study include METIS [18], LDG [15], DBH [11], and Random [8].

- **METIS** is one of the graph partitioning algorithms with the highest precision and is widely used in distributed graph computing systems. However, it is difficult for METIS to manage large graphs because it is an offline graph partitioning algorithm [14].
- **LDG** is a classic streaming graph partitioning algorithm that is characterized by the ability to quickly perform graph partitioning operations based on partial graph data information. Compared with METIS, LDG is less accurate.
- **DBH** mainly focuses on the skewed degree distribution of power-law graphs. Experiments have proved that DBH has better precision when dividing graph data that obey the power-law distribution.
- **Random** is a simple random graph partitioning algorithm. It can quickly divide a graph into different partitions in scenarios where accuracy is not required. To intuitively compare the accuracy of each algorithm, the result for Random is regarded as the worst partitioning quality in this study.

Our proposed TLP algorithm and the above four algorithms were run on nine different graph datasets with partition number p=10, 15, 20. The results are shown in Fig. 8.

Fig. 8(a), Fig. 8(b), and Fig. 8(c) show the graph partitioning results when p=10, 15, 20, respectively. The x-axis represents RF, which is a measurement of graph partitioning performance. The smaller the RF, the better the graph partitioning performance. From comparing the experimental results, we obtain the following conclusions.

- The qualities of graph partitioning with TLP and METIS are better than other algorithms in all cases.
- In most cases, TLP performs better than METIS, while some performances of TLP are slightly worse than those of METIS.

To compare TLP with METIS in more detail, the differences in RF between the two algorithms, ΔRF, is defined as follows.

$$\Delta RF = RF(\text{METIS}) - RF(\text{TLP}) \tag{12}$$

Where ΔRF >0, the performance of TLP is better than METIS. The ΔRF in all cases is shown in Table IV.

TABLE IV. VALUE OF ΔRF BASED ON NINE GRAPH DATASETS WHEN P=10, 15, 20

	G_1	G_2	G_3	G_4	G_5
p=10	1.19	0.32	0.23	−0.09	0.14
p=15	1.29	0.64	0.18	−0.07	0.08
p=20	1.56	0.85	0.20	−0.09	0.12
	G_6	**G_7**	**G_8**	**G_9**	**Average**
p=10	0.04	0.10	0.37	0.02	0.26
p=15	0.03	0.07	0.47	0.03	0.30
p=20	0.05	0.09	0.48	0.03	0.36

As shown in Table IV, ΔRF >0 in eight graphs when p=10, 15, 20, respectively, which means that the partitioning qualities of TLP are better than those of METIS in most situations. The averages of ΔRF are also larger than 0 for different values of p, thus TLP performs better than METIS overall.

C. Comparison of Different Divisions of Two Stages

To prove the superiority of the TLP algorithm that divides the two stages based on modularity, we set the division between the two stages according to the number of edges in each partition and compare it with the TLP algorithm through experiments. The change in the division between the two stages is shown in Table V.

TABLE V. THE DIVISION BETWEEN TWO STAGES BASED ON THE NUMBER OF EDGES

Stage	Criteria
Stage I	$0<\|E(P_k)\|\le R\cdot C$
Stage II	$R\cdot C<\|E(P_k)\|\le C$

where R is the ratio parameter of the two stages. $|E(P_k)|\le R\cdot C$ and $R\cdot C<|E(P_k)|$ refer to Stage I and Stage II of the graph partitioning, respectively. Particularly, there is only Stage II in the graph partitioning process when R=0, and there is only Stage I when R=1. For ease of recollection, we call this algorithm TLP_R.

To evaluate the influence of R on graph partitioning performance in detail, 11 different values of R are taken from [0, 1] with an even step length of 0.1. The experimental results with different values of R on 9 graphs are shown in Fig. 9, Fig. 10, and Fig. 11, where p=10, 15, 20.

The above insets represent the performances of TLP and TLP_R based on different real-world graphs where p=10, 15, 20. The horizontal ordinate and vertical coordinate of each inset represent different values of R and RF, respectively. Therefore, the following conclusions can be drawn from the above experimental results:

(1) In TLP_R, all the values of R satisfy $R \in (0, 1)$, corresponding to the optimal partitioning results.

(2) In TLP_R, the values of R satisfy $R \in \{0, 1\}$,

corresponding to the worst partitioning results.

(3) In TLP_R, the optimal partitioning result corresponds to different R values in different cases.

(4) Compared with TLP_R, TLP can obtain near-optimal partitioning results in most cases.

Combining conclusions (1) and (2), we deduce that graph partitioning with the two-stage heuristic method results in better quality than with the one-stage heuristic. Conclusions (3) and (4) mean TLP can obtain better partitioning results than TLP_R without adjusting parameters.

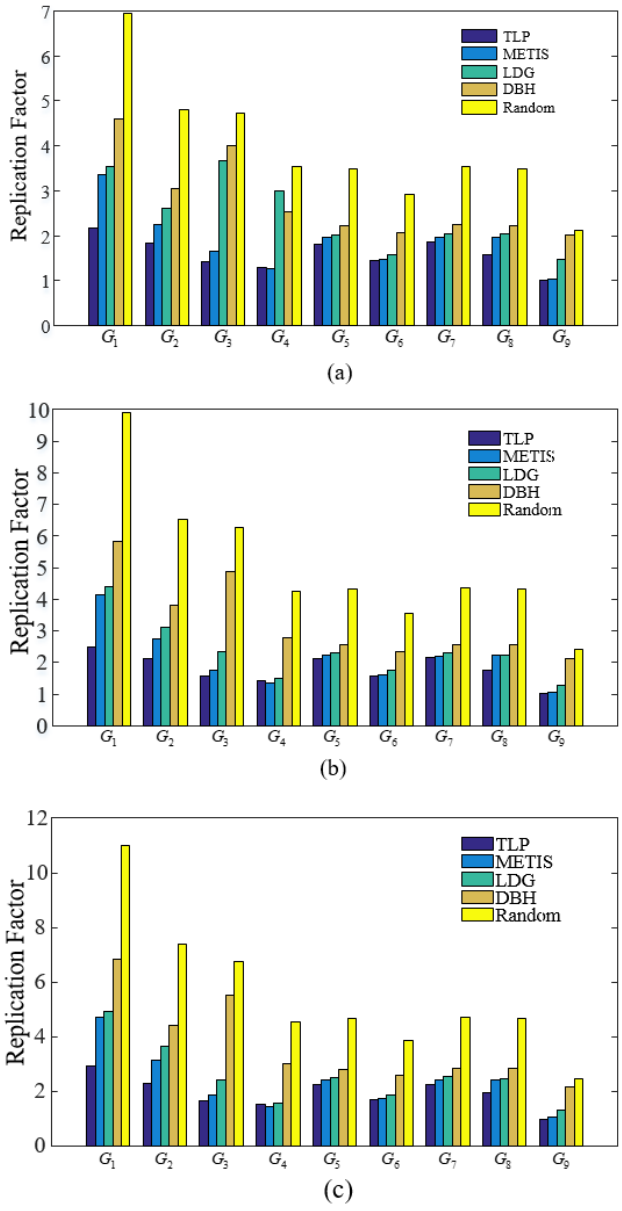

Fig. 8. Replication factors for different algorithms run on real-world graphs. The number of partitions in (a), (b), and (c) are 10, 15, and 20, respectively.

D. Analysis of Average Degree in Two-Stage Method

In this subsection, we will analyze the differences between these two stages using the average degree. As described in Section III, TLP divides the process of graph partitioning into two stages, in which different partitioning strategies are adopted. In Stage I, the core vertexes with the large degrees in the graph are more likely to be chosen. In Stage II, based on these center nodes, the local partition expands by selecting the vertices which are close to the existing partition.

To analyze the above two strategies, the average degree of all vertices in these two stages are counted. As shown in Table VI. The average degrees of all vertices in stage I are much greater than that in stage II, which means the core vertices with the large degrees are chosen in stage I indeed. In stage II, the local partition expands with these core vertices as the center, so the average degrees in stage II are smaller than that stage I.

TABLE VI. THE AVERAGE DEGREE OF ALL VERTICES IN TWO STAGES BASED ON NINE GRAPHS WHERE P=10, 15, 20

	p=10		p=15		p=20	
	Stage I	**Stage II**	**Stage I**	**Stage II**	**Stage I**	**Stage II**
G_1	45.85	13.16	32.44	11.15	39.20	9.29
G_2	57.13	7.67	46.90	6.25	63.34	7.19
G_3	236.86	12.72	196.17	12.78	123.01	12.48
G_4	31.46	7.63	41.08	6.22	29.21	5.79
G_5	33.25	8.92	34.55	8.36	30.37	8.00
G_6	65.62	5.09	78.16	5.80	46.46	5.52
G_7	48.60	9.23	26.27	8.82	32.49	8.10
G_8	13.93	10.95	45.42	9.17	53.56	8.24
G_9	30.91	4.15	166.99	4.11	58.66	4.02

V. CONCLUSIONS

In distributed graph computation systems, graph data partitioning impacts the communication overhead and the workload balance between computing resources. In this paper, a local graph partitioning method that relies on local graph information and needs to save data for only one single partition was proposed. Our proposed algorithm was tested on several real-world datasets and the results were compared with several state-of-the-art graph partitioning algorithms. The experiments demonstrated the superiority of our algorithm.

Our algorithm can be further improved in several aspects. Firstly, although the TLP algorithm can achieve good partitioning quality relying on local graph information, the time complexity of TLP is higher than some state-of-the-art graph partitioning algorithms. We expect the partitioning efficiency of TLP will be improved in the future.

Secondly, the graph data must be traversed in BFS (Breadth First Search) order when each partition expands, which means the unpartitioned graph data need to be sorted in BFS order after one vertex is partitioned. In future work, a sliding window mechanism will be introduced to sort and partition the graph data in parallel, which will be more suitable for large-scale graph partitioning.

VI. ACKNOWLEDGMENTS

This work was partly supported by the National Key Research and Development Program of China, under grant 2016YFB1000901 and the National Natural Science Foundation of China under grant 91746209. Chenyang Bu was also partly supported by the Fundamental Research Funds for the Central Universities (No. JZ2018HGBH0279), the National Natural Science Foundation of China (No. 61573327), and the Project funded by the China Postdoctoral Science Foundation (No. 2018M630704).

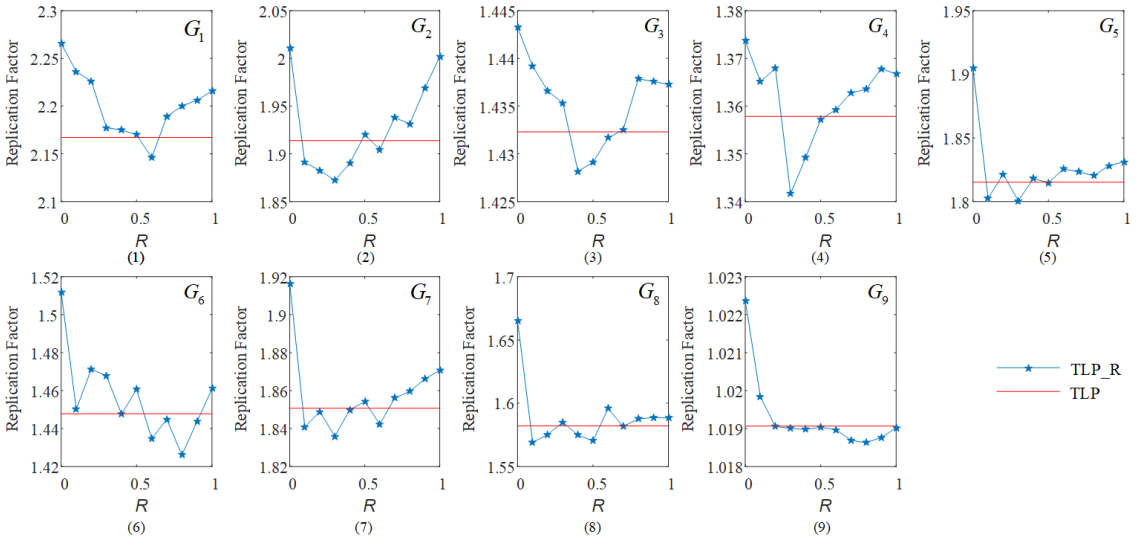

Fig. 9. Replication factors for TLP and TLP_R on real-world graphs where p=10. The values of R in TLP_R are taken from [0,1] with an even step length of 0.1. (1)-(9) is the experimental results based on $G_1 - G_9$.

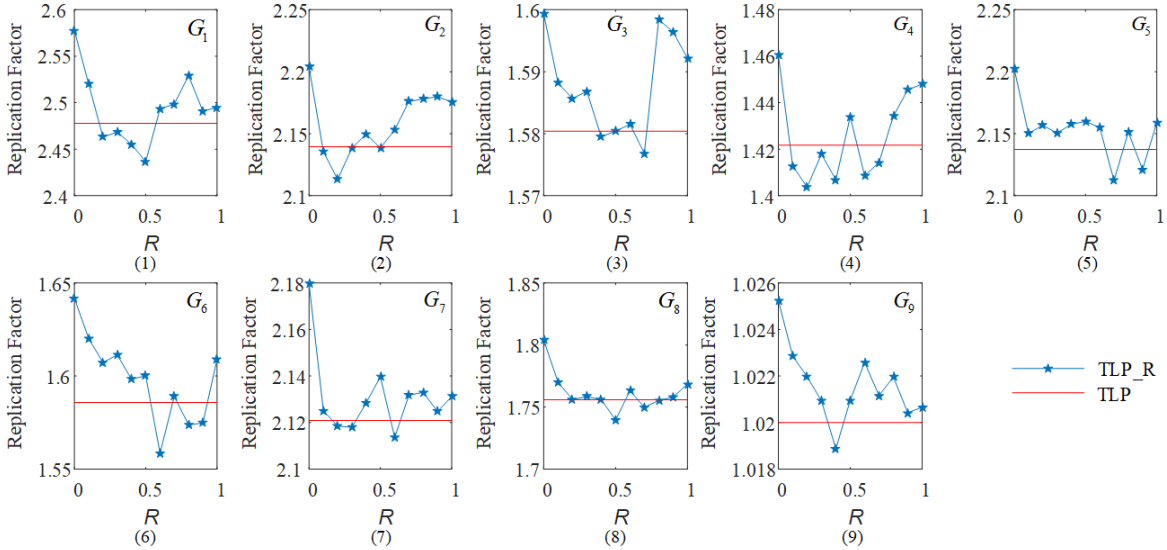

Fig. 10. Replication factors for TLP and TLP_R on real-world graphs where p=15. The values of R in TLP_R are taken from [0,1] with an even step length of 0.1. (1)-(9) is the experimental results based on $G_1 - G_9$.

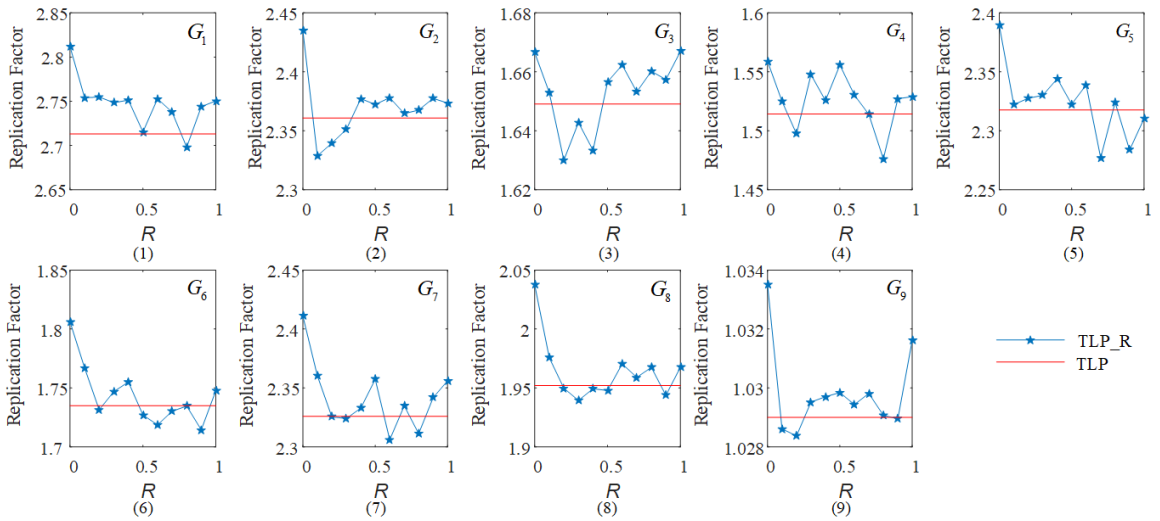

Fig. 11. Replication factors for TLP and TLP_R on real-world graphs where p=20. The values of R in TLP_R are taken from [0,1] with an even step length of 0.1. (1)-(9) is the experimental results based on $G_1 - G_9$.

REFERENCES

[1] Z. Bu, J. Cao, H. Li, G. Gao, and H. Tao, "Gleam: A Graph Clustering Framework based on Potential Game Optimization for Large-scale Social Networks," Knowledge and Information Systems, vol. 55, no. 3, pp. 741-770, 2018.

[2] W. Luo, Z. Yan, C. Bu, and D. Zhang, "Community Detection by Fuzzy Relations," IEEE Transactions on Emerging Topics in Computing, vol. 1, no. 1, pp. 1-14, 2017.

[3] W. Gao, W. Luo, and C. Bu, "Evolutionary Community Discovery in Dynamic Networks Based on Leader Nodes," in International Conference on Big Data and Smart Computing (BigComp), Hong Kong, China, 2016, pp. 53-60: IEEE.

[4] Y. Jiang, G. Wu, C. Bu, and X. Hu, "Chinese Entity Relation Extraction Based on Syntactic Features," in IEEE International Conference on Big Knowledge (ICBK), Singapore, Singapore, 2018, pp. 99-105: IEEE.

[5] Y. Shan, C. Bu, X. Liu, S. Ji, and L. Li, "Confidence-Aware Negative Sampling Method for Noisy Knowledge Graph Embedding," in 2018 IEEE International Conference on Big Knowledge (ICBK), Singapore, Singapore, 2018, pp. 33-40: IEEE.

[6] X. Wu, X. Zhu, G. Wu, and W. Ding, "Data Mining with Big Data," IEEE Transactions on Knowledge and Data Engineering, vol. 26, no. 1, pp. 97-107, 2014.

[7] G. Malewicz et al., "Pregel: A System for Large-scale Graph Processing," in ACM International Conference on Management of Data (SIGMOD), Indianapolis, Indiana, USA, 2010, pp. 135-146: ACM.

[8] J. E. Gonzalez, Y. Low, H. Gu, D. Bickson, and C. Guestrin, "Powergraph: Distributed Graph-parallel Computation on Natural Graphs," in USENIX Conference on Operating Systems Design and Implementation (OSDI), Hollywood, CA, USA, 2012, pp. 17-30: USENIX Association.

[9] Y. Low, J. E. Gonzalez, A. Kyrola, D. Bickson, C. E. Guestrin, and J. Hellerstein, "GraphLab: A New Framework for Parallel Machine Learning," in Annual Conference on Uncertainty in Artificial Intelligence (UAI), Catalina Island, CA, USA, 2014, pp. 340-349: AUAI.

[10] J. E. Gonzalez, R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, and I. Stoica, "Graphx: Graph Processing in A Distributed Dataflow Framework," in USENIX Conference on Operating Systems Design and Implementation (OSDI), Berkeley, CA, USA, 2014, pp. 599-613: USENIX Association.

[11] C. Xie, L. Yan, W. Li, and Z. Zhang, "Distributed Power-law Graph Computing: Theoretical and Empirical Analysis," in Advances in Neural Information Processing Systems (NIPS), Montreal, Quebec, Canada, 2014, pp. 1673-1681: MIT.

[12] A. Roy, L. Bindschaedler, J. Malicevic, and W. Zwaenepoel, "Chaos: Scale-out Graph Processing from Secondary Storage," in Symposium on Operating Systems Principles (SOSP), Monterey, California, 2015, pp. 410-424: ACM.

[13] C. Zhang, F. Wei, Q. Liu, Z. G. Tang, and Z. Li, "Graph Edge Partitioning via Neighborhood Heuristic," in International Conference on Knowledge Discovery and Data Mining (ACM SIGKDD), Halifax, NS, Canada, 2017, pp. 605-614: ACM.

[14] Y. Guo, S. Hong, H. Chafi, A. Iosup, and D. Epema, "Modeling, Analysis, and Experimental Comparison of Streaming Graph-partitioning Policies," Journal of Parallel and Distributed Computing, vol. 108, no. 1, pp. 106-121, 2017.

[15] I. Stanton and G. Kliot, "Streaming Graph Partitioning for Large Distributed Graphs," in International Conference on Knowledge Discovery and Data Mining (ACM SIGKDD), Beijing, China, 2012, pp. 1222-1230: ACM.

[16] F. Bourse, M. Lelarge, and M. Vojnovic, "Balanced Graph Edge Partition," in International Conference on Knowledge Discovery and Data Mining (ACM SIGKDD), New York, USA, 2014, pp. 1456-1465: ACM.

[17] B. W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs," The Bell System Technical Journal, vol. 49, no. 2, pp. 291-307, 1970.

[18] G. Karypis and V. Kumar, "A Fast And High Quality Multilevel Scheme for Partitioning Irregular Graphs," SIAM Journal On Scientific Computing, vol. 20, no. 1, pp. 359-392, 1998.

[19] D. Margo and M. Seltzer, "A Scalable Distributed Graph Partitioner," in International Conference on Very Large Data Bases (VLDB Endowment), Kohala Coast, Hawaii, 2015, vol. 8, no. 12, pp. 1478-1489: ACM.

[20] C. Tsourakakis, C. Gkantsidis, B. Radunovic, and M. Vojnovic, "Fennel: Streaming Graph Partitioning for Massive Scale Graphs," in ACM International Conference on Web Search and Data Mining (WSDM), New York, USA, 2014, pp. 333-342: ACM.

[21] F. Luo, Y. Yang, C. Chen, R. Chang, J. Zhou, and R. H. Scheuermann, "Modular Organization of Protein Interaction Networks," Bioinformatics, vol. 23, no. 2, pp. 207-214, 2006.

[22] W. Luo, D. Zhang, H. Jiang, L. Ni, and Y. Hu, "Local Community Detection with the Dynamic Membership Function," IEEE Transactions on Fuzzy Systems, vol. 26, no. 5, pp. 3136-3150, 2018.

[23] J. Cao, Z. Bu, G. Gao, and H. Tao, "Weighted Modularity Optimization for Crisp and Fuzzy Community Detection in Large-scale Networks," Physica A: Statistical Mechanics and its Applications, vol. 462, no. 15, pp. 386-395, 2016.

[24] J. Leskovec and R. Sosič, "Snap: A General-purpose Network Analysis and Graph-mining Library," ACM Transactions on Intelligent Systems and Technology, vol. 8, no. 1, pp. 1-20, 2016.

[25] The website of huapu system. (2017). Accessed on May 15 2017. [Online]. Available: http://huapu.bigke.org/