Clustering-Guided Dynamic Social Network Graph Partitioning

Jin Li[1], Peiyuan Chen[1(✉)], Zhizhan Shen[1], Jianhua Cheng[1], and Feng Zhang[1]

Harbin Engineering University, Harbin, Heilongjiang 150001, P.R. China
{lijinokok,peiyuanchen,zhizhan_shen,ins_cheng,zhfe981206}@hrbeu.edu.cn

Abstract. Graph partitioning is a critical step in distributed computing for large-scale social network graphs, aiming to divide the graph into balanced partitions for downstream tasks. The structure and scale of real-world social network graphs continuously evolve, impacting the quality of graph partitions and, consequently, the performance of downstream tasks. Most Existing methods adopt batch incremental partitioning or repartitioning strategies to process dynamic changes, but they often lack real-time processing capabilities and struggle to optimize partition quality. To mitigate these challenges, we propose a clustering-guided dynamic social network graph partitioning method(CG-DGP). In the first stage, we use streaming community detection combined with modularity evaluation to extract clustering information from the social network, enabling tightly connected vertices are allocated to the same partition. In the second stage, dynamic changes are processed in real time via edge streaming, applying different partitioning strategies for high- and low-degree edges, while optimizing partition quality in real time by reassigning edge groups. Experimental results demonstrate that CG-DGP efficiently generate the-state-of-the-art replication factor while maintaining load balance. The runtime for the downstream tasks of PageRank and connected components can be reduced by up to 9.68% and 10.89%, respectively.

Keywords: Social network · Dynamic graph partitioning · Cluster information · Community detection · edge groups.

1 Introduction

In the past decade, relationship data in social networks have grown exponentially, forming complex social network graphs composed of individuals, groups, and organizations. These graphs are invaluable resources in areas such as public opinion analyses, link predictions, and personalized recommendations. With the advancement of internet technology, the scale of real-world graphs has rapidly expanded, necessitating the execution of graph storage and processing tasks in distributed environments. This has led to the emergence of numerous distributed graph processing systems,such as [1–3].

As a foundational task in distributed graph processing, graph partitioning aims to create balanced partitions while minimizing inter-partition communication overhead [4]. However, balanced graph partitioning is an NP-hard problem [5]. In recent years, research has shifted from vertex partitioning to edge

partitioning, which is more suitable for large-scale power-law graphs like social networks [3, 6]. In edge partitioning, each edge is assigned to a partition, allowing neighboring edges of the same vertex to reside in different partitions. This approach creates vertex replicas in related partitions, effectively addresses load imbalance caused by supernodes. Because distributed computation typically operates along edges, the partition load is determined by the number of edges. To reduce the communication overhead of synchronizing replicas, edge partitioning seeks to minimize vertex replicas while maintaining balanced partition loads.

Many advanced edge partitioning algorithms have been proposed, yet most start by partitioning an entire graph from scratch. In real-world scenarios, social network graphs are dynamic and evolve over time, with increasing density [7]. Dynamic changes result in tighter connections between partitions, degrading partition quality and affecting distributed computation performance. Some studies address dynamic changes in batches [8–10], though these approaches are not suitable for real-time applications. Other methods employ streaming partitioning for real-time edge addition [11–15], but they often fail to handle edge deletions and may have lower partition quality. Re-partitioning methods aim to optimize partition quality after changes [16–18], however, their performance and optimization results are limited.

To address these challenges, we propose a **C**lustering-**G**uided **D**ynamic **G**raph **P**artitioning method (CG-DGP) for dynamic social network graph partitioning. We leverage the cluster information within social network, utilizing community detection and evaluating modularity to obtain high-quality initial partition results. For dynamic changes, we design an edge-degree-aware partitioner that adopts differentiated strategies for high- and low-degree edges in dynamic changes. Furthermore, to mitigate the partition quality degradation caused by dynamic changes, we enhance the graph structure optimizer in the graph re-partitioning algorithm GR-DEP [18] by using a greedy strategy for edge groups search and redesigning the priority scoring function. This approach achieves lower time and space complexity while improving the success rate of the search, thereby enhancing the overall quality of dynamic graph partitioning.

The main contributions of this study are as follows:

- We propose a community-aware streaming graph partitioning algorithm that leverages the cluster information within social network graphs. By streaming community detection and evaluating modularity, we ensure full utilization of cluster information, thereby achieving high-quality initial partitioning of dynamic social network graphs.
- We design an edge-degree-aware dynamic change addressing algorithm that processes dynamic changes in real time by streaming edge input. This method adopts differentiated strategies for high- and low-degree edges and optimizes partition quality degradation caused by dynamic changes using an enhanced graph structure optimizer.
- We evaluate CG-DGP on four real-world social network datasets of different scales. The results show significant performance improvements over state-

of-the-art dynamic graph partitioners in two distributed graph processing tasks: PageRank and connected components.

2 Related Work

Dynamic graph partitioning has seen diverse approaches aimed at addressing dynamic changes and improving partition quality. Main methods include streaming algorithms, incremental algorithms, and graph re-partitioning methods.

Streaming graph partitioning algorithms, originally designed for static graphs, allocate vertices or edges individually with low overhead, making them applicable to dynamic graphs. However, these algorithms often face the "uninformed allocation" problem [11], where early decisions lack awareness of future graph structure, leading to suboptimal results. Enhanced methods such as Restreaming [12], LEOPARD [13], and Adwise [14] leverage global or local topology to improve partitioning. Despite their enhancements, these approaches typically require operations on the entire graph or delayed processing of dynamic changes, limiting their suitability for real-time applications.

Incremental graph partitioning algorithms, including IncDBH [8], IncHDRF [9], and DynamicDFEP [10], focus on efficiently integrating new edges. These methods extend existing frameworks to handle dynamic changes but often struggle with real-time processing and partition quality optimization.

Graph re-partitioning methods, such as Hermes [16], GrapH [17], and GR-DEP [18], optimize partition quality by periodically adjusting partitions. These methods aim to reduce communication overhead and improve load balance but often require significant computational resources.

By contrast, our proposed CG-DGP leverages clusters from social network graphs to achieve high-quality partitions while processing dynamic changes in real-time and optimizing partition quality.

3 Preliminaries

Definition 1. *(Dynamic Graph Partition). Let the graph be* $G = (V, E)$*, where* V *and* E *represent the sets of vertices and edges, respectively. At each timestamp* t_i *in the time series* $t = \{t_1, t_2, \ldots, t_{n-1}\}$*, a new graph* $G_i = (V_i, E_i)$ *is generated, where:* $V_i = V_{i-1} \cup V_{Ii} - V_{Di}$,$E_i = E_{i-1} \cup E_{Ii} - E_{Di}$.$V_{Ii}$ *and* E_{Ii} *denote the vertices and edges added at timestamp* t_i*, while* V_{Di} *and* E_{Di} *denote the vertices and edges removed at* t_i*.* ΔG_i *represents the dynamic changes in the graph data during this period. Dynamic graph partitioning involves designing strategies to handle the dynamic changes* ΔG_i *based on an initial partition P, resulting in an updated partition* $P_i = \{P_{i1}, P_{i2}, \ldots, P_{ik}\}$*. The goal is to balance the edge counts across partitions while minimizing the vertex cut* $VC(P)$*.Let* $\mathrm{R(v)}$ *represent the set of partitions covering vertex* $\mathrm{v} \in \mathrm{V}$*, and* $|\mathrm{R(v)}|$ *denote the number of replicas of vertex* v*. The objectives of dynamic graph partitioning include minimizing the average number of vertex replicas (replication factor* $\mathrm{RF(P)}$*) and reducing the*

differences between partition sets across timestamps. The problem is formalized as follows:

$$\min RF(P) = \frac{\sum_{v \in V} |R(v)|}{|V|} s.t. \quad |P_i| \leq \varepsilon \frac{|E|}{k} (1 \leq i \leq k) \tag{1}$$

Here, The imbalance factor ε satisfies $\leqslant \varepsilon \leqslant 2$. When $\varepsilon = 1$, the number of vertices in each partition must be strictly equal. When $\varepsilon = 2$, the number of vertices in each partition is allowed to be up to twice the average.

Definition 2. *(External Vertex-Cut, EVC). Given an edge set $S_i \subseteq P_i$, a locally suboptimal partition $LSP(S_i, P_j)$ with respect to partition P_j, and edge groups $EG \subseteq S_i$, the vertices in $V(EG) \cap VC(S_i, P_j)$ are referred to as EVC.*

Definition 3. *(Internal Vertex-Cut ,IVC). Given an edge set $S_i \subseteq P_i$, a locally suboptimal partition $LSP(S_i, P_j)$ with respect to partition P_j, and edge groups $EG \subseteq S_i$, the vertices in $V(EG) \cap V(P_i \setminus EG)$ are referred to as IVC.*

Definition 4. *(Migration Gain). For edge groups $EG \subseteq P_i$, the migration gain of transferring EG from P_i to P_j is defined as the reduction in the number of vertex replicas, as shown in Eq. 2.*

$$gain_{i \to j}(EG) = \left|EVC\right| - \left|IVC\right| \tag{2}$$

4 Method

To achieve real-time and high-quality partitioning, we propose CG-DGP that leverages the widely existing cluster information in social network graphs. The method consists of two stages: (1) community-aware streaming graph partitioning, which utilizes streaming community detection and modularity evaluation to fully exploit clusters within graphs, thereby reduce replication factors, and achieve high-quality partitioning; (2) edge-degree-aware dynamic change processing, which applies differentiated strategies for high- and low-degree edges and uses a graph structure optimizer for real-time partition optimization. The framework is depicted in Figure 1, where three distinct clusters are represented by three different colors, and bold dashed lines indicate overloaded edges and cross-partition cut edges.

4.1 Community-Aware Streaming Graph Partitioning

Clusters of community structures in social networks, characterized by dense internal and sparse external connections, can effectively enhance the quality of graph partitioning [19]. To this end, we adopt a community-aware streaming graph partitioning method (Figure 1(a)). First, community structure is detected through streaming community detection, then mapped to partitions basing on modularity evaluation to fully utilize the community structure. Subsequently, edges between vertices within partitions are preassigned, while overloaded edges and cut edges are handled by a community-aware edge partitioning algorithm, completing the initial partitioning of the dynamic social network graph.

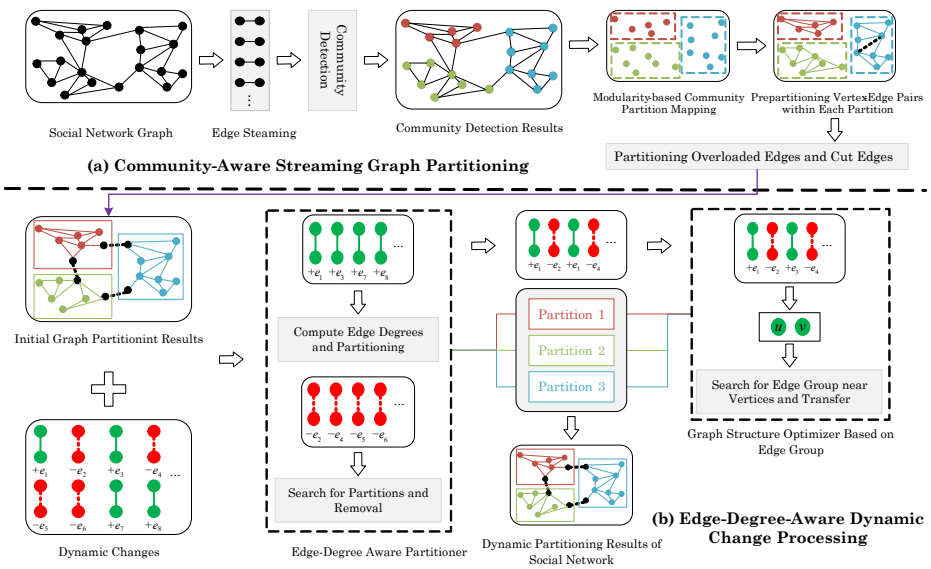


Fig. 1: The overall architecture of the proposed method CG-DGP.

Streaming Community Detection The streaming community detection algorithm identifies community structures by scanning edges in the graph in a single pass, without loading the entire graph, considerably reducing memory usage and computational complexity, making it ideal for large-scale social networks.

The algorithm processes an edge stream as input and assigns community memberships to vertices. As noted by Hollocu et al., community structures typically exhibit high internal connectivity, and edges are more likely to be found within communities rather than between them [20]. For any edge $e = (u, v)$ in the dynamic graph, the contribution of the edge to different communities is assessed by considering the proportion of neighbors of vertices u and v within their respective communities relative to the total number of neighbors of each vertex. Based on this contribution score, the edge e is assigned to the community with stronger connectivity. This process is repeated iteratively until all edges have been assigned, thereby completing the community detection. The "community volume" is defined as the sum of the degrees of all vertices within a community, which quantifies its size. The detailed steps are presented in Algorithm 1.

Modularity-based Community Partition Mapping The number of communities generated by the streaming community detection algorithm typically exceeds the number of partitions. Therefore, during the mapping process, it is crucial to preserve community structures as much as possible to avoid information loss. Modularity, a metric for evaluating community quality, reflects the density of connections within a community [21]. Modularity-based Community Partition Mapping algorithm balances partition loads while maximizing partition

Algorithm 1 Streaming Community Detection

Input:
d ▷ vertex degree;
k ▷ number of partitions;
$N(v)$ ▷ neighbors of v;
$N_{C_i}(v)$ ▷ neighbors of v in community C_i;
max_vol ▷ maximum community volume;
Output:
$v2c$ ▷ vertex to community mapping;
$vols$ ▷ community volumes;

$max\_vol \leftarrow \frac{2*|E|}{k} \times 0.5$
for $e(u,v) \in edge\_stream$ **do**
  for $vtx \in \{u,v\}$ **do**
    if $v2c[vtx] = NULL$ **then**
      $v2c[vtx] \leftarrow new\_id$
      $vols[new\_id] \leftarrow d[vtx]$
      $new\_id \leftarrow new\_id + 1$
  if $vols[v2c[v]] \leq max\_vol$ **and** $vols[v2c[u]] \leq max\_vol$ **then**
    $Con(u) \leftarrow \frac{N_{C_u}(u)}{N(u)}$
    $Con(v) \leftarrow \frac{N_{C_v}(v)}{N(v)}$
    if $Con(u) \leq Con(v)$ **and** $vols[v2c[v]] + d[u] \leq max\_vol$ **then**
      $vols[v2c[u]] \leftarrow vols[v2c[u]] - d[u]$
      $vols[v2c[v]] \leftarrow vols[v2c[v]] + d[u]$
    if $Con(v) \leq Con(u)$ **and** $vols[v2c[u]] + d[v] \leq max\_vol$ **then**
      $vols[v2c[v]] \leftarrow vols[v2c[v]] - d[v]$
      $vols[v2c[u]] \leftarrow vols[v2c[u]] + d[v]$
return $v2c$, $vols$.

modularity, thereby preserving community integrity and improving the quality of graph partitioning.

The algorithm takes the results of community detection as input and outputs a mapping of communities to partitions. Following the principle of modularity maximization, the algorithm processes each community iteratively to determine its optimal partition assignment. Each community is evaluated against all potential partitions, and calculates the modularity for each assignment.The assignment that maximizes global modularity is selected. This process is repeated until all communities are mapped. The detailed procedure is outlined in Algorithm 2.

Community-aware Edge Partitioning The community-aware edge partitioning algorithm assigns suitable partitions for overloaded edges and cross-partition cut edges. Leveraging the local connectivity characteristics of communities, edges are more likely to be allocated to the partitions of their two vertices, while the probability of vertex replication across other partitions is lower. This allows the scoring process to consider at most two partitions, significantly reducing computational overhead. By utilizing the global structure information

Algorithm 2 Modularity-based Community Partition Mapping

Input:
v2c ▷ vertex_id to community_id mappings;
vols ▷ community volumes;
vol_p ▷ partition volume;
max_vol_p ▷ maximum partition volume;
Output:
c2p ▷ community_id to partition_id mapping;

$max\_vol\_p \leftarrow \frac{2*|E|}{k} \times 0.5$
$s\_coms \leftarrow v2c$ sorted by $vols$(descending)
for $c\_v\_pair \in s\_coms$ **do**
  $mod\_p \leftarrow$ modularity(assign c_v_pair to each partition)
  $s\_mod\_p \leftarrow$ sorted mod_p(descending)
  for $mod\_p\_pair \in s\_mod\_p$ **do**
    if $vol\_p[mod\_p\_pair.index] < max\_vol\_p$ **then**
      $p \leftarrow mod\_p\_pair.index$
      break
  assign c_v_pair to partition p , update the vol_p and $c2p$
return $c2p$

from community detection, the algorithm addresses the issue of random allocation caused by empty partition states in traditional streaming algorithms. The scoring function incorporates community volume, prioritizing edges associated with larger communities to optimize partitioning results.

The algorithm takes an edge stream as input and combines partition states with community information to allocate partitions for overloaded and cross-partition cut edges, generating an initial partitioning of the dynamic social network graph. The process includes the following steps:(1)Read the edge stream: Process only unassigned overloaded and cut edges, skipping edges already assigned.(2)Evaluate partition load: For each edge, check the load status of the partitions where its two vertices reside: If a partition is overloaded, allocate the edge using a vertex-degree-based hashing strategy.If not overloaded, traverse the partitions of the two vertices, calculate scores based on the scoring function, and assign the edge to the partition with the highest score.(3)Iterate until all edges are processed, producing the final initial partitioning result.

The scoring function evaluates the suitability of assigning an edge to different partitions, consisting of a replication factor score $g_{\{u,v\}}$ and a community score $sc_{\{u,v\}}$ (Eq. 3). The input includes the edge to be assigned, the target partition's state, and community information, and the output is the edge's score for the partition.

$$s(u, v, p) = g_{\{u,v\}} + sc_{\{u,v\}} \tag{3}$$

The replication factor score (Eq. 4) prioritizes the partitioning of high-load vertices to reduce the global replication factor. If both vertices of an edge are in the partition, the score is the maximum value of 3; if neither is in the partition,

the score is 0; if only one vertex is in the partition, the score ranges from 0 to 3, with a tendency to assign the edge to the partition of the vertex with the higher degree, thus reducing replication factor.

$$g_{\{u,v\}} = \begin{cases} 1 + \left(1 - \frac{d_{\{u,v\}}}{d_u + d_v}\right) & \text{if} \{\mathbf{u}, \mathbf{v}\} \text{is replicated on p} \\ 0 & \text{else.} \end{cases} \tag{4}$$

The community score (Eq. 5) utilizes community information to avoid random assignment issues. If the communities of both vertices of an edge are in the partition, the score is the maximum value of 1; if neither community is in the partition, the score is 0; if only one community is in the partition, the score ranges from 0 to 1, with a tendency to assign the edge to the partition of the vertex with the larger community volume. This increases the probability of subsequent edge flows being hit, further reducing the replication factor.

$$sc_{\{u,v\}} = \begin{cases} \frac{\mathrm{vol}\left(c_{\{u,v\}}\right)}{\mathrm{vol}\left(c_u\right) + \mathrm{vol}\left(c_v\right)} & \text{if} c_{\{u,v\}} \text{is assigned to p} \\ 0 & \text{else.} \end{cases} \tag{5}$$

4.2 Edge-Degree-Aware Dynamic Change Processing

The dynamic nature of social networks is primarily characterized by the internal changes of community structures. This section introduces an edge-degree-aware dynamic change processing algorithm based on the community-aware streaming graph partitioning algorithm to address dynamic changes. As shown in Figure 1(b), the algorithm consists of an edge-degree-aware partitioner and a graph structure optimizer based on edge groups. The process is as follows: (1) load the initial partition results of the dynamic graph; (2) input dynamic changes in the form of edge streams into the partitioner for processing; (3) after processing, the optimizer searches and transfers group edges within the local graph structure to improve the partition quality; the dynamic graph partitioning is then completed.

Edge-degree-aware Partitioner The edge-degree-aware partitioner applies the HDRF strategy to high-degree edges and the greedy strategy to low-degree edges in dynamic changes, assigning the changes to appropriate partitions to minimize the impact on the initial partitioning. The partitioner takes dynamic changes as input, outputs the processing strategy, and passes the results to the group-edge-based graph structure optimizer. The algorithm processes dynamic changes sequentially, designing separate strategies for deletion and addition, and applying corresponding method based on the degree of the new edges. While processing the current changes, the results are handed over to the graph structure optimizer for local optimization. The detailed process is shown in Algorithm 3.

Algorithm 3 Edge-degree-aware Partitioner

Input:
P ▷ partitions;
max_load, min_load ▷ load information;
deg_u, deg_v ▷ vertex degrees;
deg_avg ▷ global average degree;
Output:
$machine\_id$ partition information;

Initialize $dynamic\_type \in \{0 : add, 1 : delete\}$
$e(u, v) \rightarrow$ dynamic changed edges
if $dynamic\_type = 0$ **then**
  $deg\_edge \leftarrow deg_u + deg_v$
  for $p \in P$ **do**
    if $deg\_edge \geq deg\_avg$ **then**
      $score \leftarrow$ HDRF score for $e(u, v)$ in p
    else
      $score \leftarrow$ Greedy score for $e(u, v)$ in p
    if $score > max\_score$ **then**
      $max\_score \leftarrow score$
      $candidate \leftarrow \{p\}$
    else if $score = max\_score$ **then**
      $candidate \leftarrow candidate \cup \{p\}$
  $machine\_id \leftarrow$ random choice from $candidate$
else
  delete $e(u, v)$ from all partitions
send $e(u, v)$ to optimizer
return $machine\_id$

Edge-group-based Graph Structure Optimizer The reassignment of edge groups is restricted to the local graph structure with low overhead. This minimizes partition discrepancies during dynamic processing, alleviating the deterioration in partition quality and improving overall performance. The graph structure optimizer optimizes the graph through local search and edge groups' reassignment, using a greedy strategy. It doesn't require backtracking or storing historical computation results, ensuring high efficiency. The design of the priority scoring function further enhances the success rate of the search, optimizing the dynamic graph partition quality.

The optimizer takes dynamic changes and partition information as input, initializing a priority queue Q, a group edge set EG, a vertex set VG, an external cut array evc, and an internal cut value ivc. The scoring function calculates vertex priority to guide the local search. When Q is not empty, the vertex v with the highest score is popped, and its edges with VG are added to EG, and v is added to VG. The evc and ivc are updated, the partition transfer gain computed, and the target partition with the highest gain selected. If the target partition is not overloaded and the gain is positive, the edge groups reassignment is completed; otherwise, if v is not a high-load vertex, its neighbors and priority

are added to Q for further iteration. The search ends when Q is empty or EG reaches its limit.

The priority scoring function plays a pivotal role in group edge optimization, guiding the graph search direction and directly influencing the success rate of edge groups search. According to Eq. 2, maximizing transfer gain requires increasing $|EVC|$ while reducing $|IVC|$.

Optimization of $|EVC|$. A higher $Priority_e(v)$ increases the likelihood that vertex v contributes to $|EVC|$. Specifically: (1) If v is a boundary vertex (with replicas in other partitions), its inclusion directly increments $|EVC|$. (2) If v is not a boundary vertex (without replicas), a higher proportion of its neighbors having replicas increases the chances of finding boundary vertices and edge groups in subsequent searches. Consequently, boundary vertices are prioritized, while non-boundary vertices serve as complementary candidates (Eq. 6).

$$Priority_e(v) = \begin{cases} |R(v)| - 1 & \text{if } v \text{ has replicas} \\ \sum_{v_i \in N(v)} \frac{R(v_i)}{|P|*|N(v)|} & \text{otherwise} \end{cases} \tag{6}$$

Optimization of $|IVC|$. Similarly, a higher $Priority_i(v)$ indicates a greater potential to reduce $|IVC|$. This evaluates two cases: (1)Fewer external neighbors: If v has fewer neighbors outside VG, its contribution to $|IVC|$ is smaller, yielding a higher score. (2)Reduction of external neighbors: The inclusion of v may reduce the number of its external neighbors, potentially removing them from $|IVC|$ in subsequent expansions. This reduction further prioritizes v (Eq. 7).

$$\begin{gathered} Priority_i(v) = Priority_{i1}(v) + Priority_{i2}(v) \\ Priority_{i1}(v) = \frac{1}{(|N(v) \setminus VG| + 1)^2}, \quad Priority_{i2}(v) = \sum_{v_i \in N(v) \cap VG} \frac{1}{|N(v_i) \setminus VG|} \end{gathered} \tag{7}$$

Combined Scoring Function. The overall priority score integrates $Priority_e(v)$ and $Priority_i(v)$, as defined in Eq. 8:

$$Priority(v) = Priority_e(v) + Priority_i(v) \tag{8}$$

This scoring framework effectively balances edge contribution and internal connectivity, enhancing edge groups optimization.

5 Theoretical Analysis

Time Complexity Analysis. The method presented in this paper comprises a community-aware streaming partitioning stage and an edge-degree-aware dynamic change processing stage. The community-aware streaming partitioning stage uses a modularity-based community detection algorithm, with a time complexity of $O(|E|)$, because it traverses the edge streaming to identify community structures and allocate edges. In the dynamic change processing stage, the edge-degree-aware partitioner manages all dynamic changes with a time complexity is

approximately $O(|P| \times |E|)$, where $|P|$ is the number of partitions and $|E|$ represents the set of edges affected by dynamic changes. By combining both stages, the time complexity of the algorithm is $O(|E| + |P| \times |E|)$, demonstrating linear scalability with the number of edges.

Space Complexity Analysis. The method stores vertex degrees, community mappings, and partition state information. The initial streaming community detection and mapping procedures require $O(|V| + |P|)$ space, where $|V|$ represents the number of vertices. During dynamic change processing, additional memory is required for the priority queues and edge groups details in local graph structure, leading to a space complexity of $O(|Q| + 2 \times |maxEG| + |P|)$, where $|maxEG|$ denotes the maximum size of the edge groupss. Overall, the space complexity of the algorithm is $O(|V| \times |P|)$, which is well-suited for the real-time processing of large-scale social network graphs.

6 Evaluation

In this section, we assess the performance of different dynamic graph partitioning algorithms on dynamic social network graphs of various scales. The evaluations were performed on a distributed graph computation platform using specialized algorithms to evaluate partitioning quality. We will start with an introduction to the datasets, baselines, and experimental setup, followed by in-depth analysis of the results.

6.1 Setup

Dataset: As shown in Table 1, four real-world social network datasets of varying scales were used in the experiments: the online social networks Facebook and Twitter [22], Orkut [23], and the comprehensive social network LiveJournal [23]. To create a dynamic environment, we employed the graph sampling strategy detailed in [24] to partition each dataset into an initial static segment and dynamically evolving segment.

Table 1: Real-world graph datasets.

Name	|V|	|E|	Communities
Facebook(FB)	4,039	88,234	193
Twitter(TW)	81,362	1,768,149	4,869
LiveJournal(LJ)	3, 997, 962	34, 681, 189	311,782
Orkut(OR)	3, 072, 441	117, 185, 083	8,455,253

Evaluation Metrics: To assess the effectiveness of the CG-DGP, the experimental evaluation focuses on the following tasks:

(1) Partition Quality: Assessing the quality of graph partitioning through replication factor and load balance.
(2) Partition Time: Measuring the time required to complete the dynamic graph partitioning task.
(3) Scalability: Analyzing the variation in the replication factor as the number of partitions increases.
(4) Application: Evaluating the impact of different graph partitioning algorithms on downstream graph computation tasks.

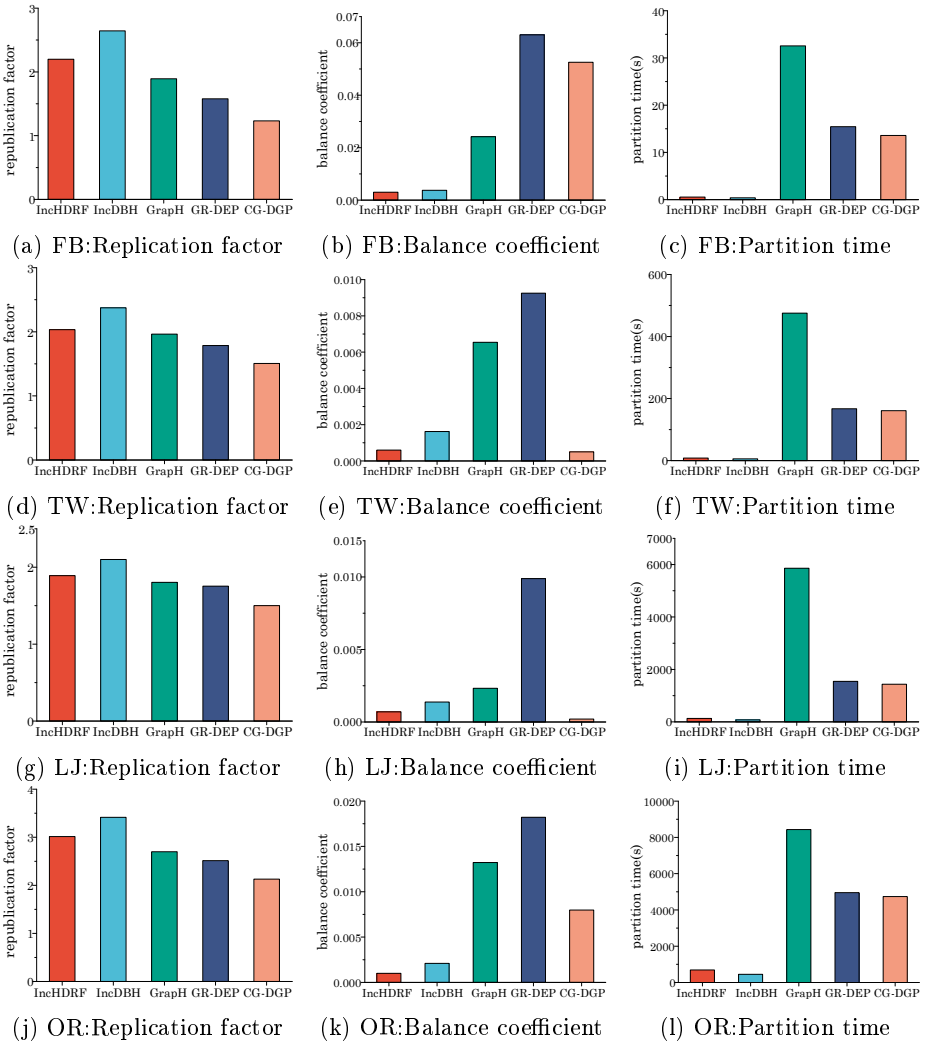
In our experiments, we used the replication factor and the relative standard deviation of load to valuate partition quality. Specifically, the average replication factor for each vertex was determined using Eq. 1. Futhermore, the balance coefficient γ was defined as the ratio of the maximum vertes load in the heaviest partition to the average vertex load across all partitions,as detailed below:

$$\gamma = \frac{max_{p_i \in P}|V(p_i)|}{\frac{1}{|P|}\sum_{p_i \in P}|V(p_i)|} - 1 \tag{9}$$

Baseline: Two real-time incremental graph partitioning algorithms, IncDBH [8] and IncHDRF [9], and two lightweight graph re-partitioning algorithms, GrapH [17] and GR-DEP [18], are selected as the control group in this study.
Experimental Setup: All experiments were conducted on a server equipped with an Intel(R) Core(TM) i9-7980XE Extreme Edition processor (4.20 GHz) and 128GB of memory. To simulate a distributed computing environment, we implemented a cluster with the following configuration: the master node operates in an unrestricted container without memory or CPU limitations, while four worker node containers are each limited to 32GB of memory and 4 CPU cores.
Parameter Settings: The number of partitions was set to 4 to accommodate subsequent distributed graph computing tasks. The imbalance factor was set to 1.03, permitting for a load imbalance of up to 3%. For dynamic changes, 30% of the edges were extracted via graph sampling, with 70% being edge additions and 30% edge deletions. The graph computing experiments used the PageRank and connected components (CC) algorithm, which were configured to a maximum of 100 iterations.

6.2 Results

Partition Quality: we assess the quality of graph partitions generated by using dynamic graph partitioning algorithms on different datasets. As illustrated in Figure 2, it can be observed that, on most datasets, the incremental algorithms produce partitions with high replication factors, which increases communication overhead and runtime during distributed graph computations. Conversely, the re-partitioning algorithms optimize local structures after processing dynamic changes, yielding improved partition quality. Notably, our proposed algorithm delivers the best partitioning results on most datasets. All dynamic graph partitioning algorithms can generate relatively balanced graph partitions i.e., balance coefficient less than 0.1.



(a) FB:Replication factor (b) FB:Balance coefficient (c) FB:Partition time

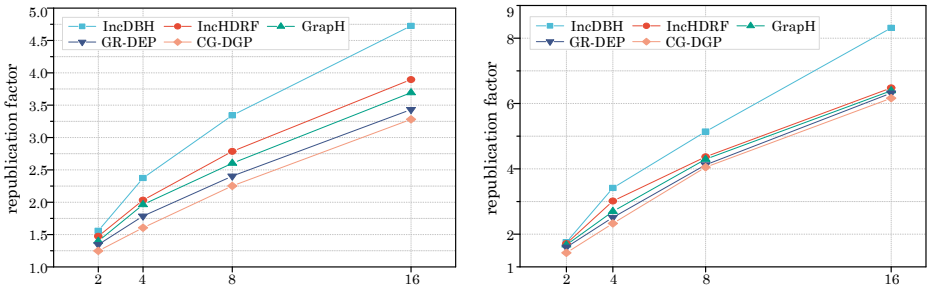
(d) TW:Replication factor (e) TW:Balance coefficient (f) TW:Partition time

(g) LJ:Replication factor (h) LJ:Balance coefficient (i) LJ:Partition time

(j) OR:Replication factor (k) OR:Balance coefficient (l) OR:Partition time

Fig. 2: Performance results on real-world graphs.

Execution Time: The purpose of graph partitioning is to enhance the performance of distributed computation. Low replication factor reduces communication overhead, thereby decreasing the running time of graph computation tasks. However, the time required for partitioning must be considered for overall performance evaluation. Figure 2 illustrates the runtime of different algorithms on social network graphs. CG-DGP exhibits the highest partition quality, its runtime is positioned third. This runtime is lower than GrapH and GR-DEP, which also use the graph re-partitioning strategy to process dynamic changes. Graph



(a) TW:Republication factor (b) OR:Republication factor

Fig. 3: The Replication factor on different datasets under multiple partitions.

re-partitioning algorithm generally requires more time to optimize local graph structures, enabling higher-quality partitions than incremental methods.

Scalability: The following insight are derived frome the results of multi-partition dynamic graph partitioning experiments performed on different datasets(Figure 3). The following conclusions can be drawn: (1) For the same dataset, replication factors increases as the number of partitions increases. (2) For a fixed number of partitions, replication factors increases as the dataset size increases. (3) CG-DGP consistently demonstrates the lowest replication factors across all dataset sizes, highlighting its high scalability.

Application: To assess the utility of partitions, we performed to graph computation tasks in GraphX. Table 2 provides a summary of the execution time(Time) and communication overhead(Comm) for these tasks across partitions generated by different algorithms on various datasets. The following conclusions can be drawn: (1) Large datasets increase the runtime and network communication in distributed graph computations. (2) Given that all dynamic graph partitioning algorithms maintain high load balance, the communication overhead is largely determined by the replication factors of partitions. (3) CG-DGP outperforms other methods in terms of the execution time and communication overhead, delivering superior performance for graph computation tasks.

7 Conclusion

In this paper, we propose CG-DGP, a clustering-guided dynamic social network graph partitioning algorithm that enables real-time and high-quality partitioning for dynamic social network graph. Our findings indicate that leveraging clusters in social networks greatly imporves partitioning quality. To utilize this structure information guiding graph partition, CG-DGP uses community detection and edge groups search. The experimental results demonstrate that CG-DGP delivers superior partition quality compared to state-of-the-art methods while requiring lower partitioning time. Furthermore, the performance of downstream tasks can be significantly enhanced by benefiting from the partitioning result.

Table 2: Time(s) and Communication overhead(GB) for graph computation.

Task	Graph	IncDBH	IncHDRF	GrapH	GR-DEP	CG-DGP
		Time	*Time*	*Time*	*Time*	*Time*
		Comm	*Comm*	*Comm*	*Comm*	*Comm*
PageRank	Facebook	23.5066	23.2611	23.0932	23.1278	**23.0105**
		1.5499	1.5500	1.5499	1.5500	**1.5408**
	Twitter	32.6654	32.2098	31.8743	31.5154	**30.7697**
		1.5503	1.5494	1.5490	1.5499	**1.5389**
	LiveJournal	284.6712	267.1386	266.0741	264.0642	**257.1275**
		26.6853	24.8923	24.1391	23.6332	**22.1897**
	Orkut	608.4312	579.1380	574.5631	573.7143	**555.4353**
		43.5187	38.6849	36.9216	35.2825	**32.8391**
CC	Facebook	9.4576	9.3819	9.3098	9.4701	**9.2563**
		1.5173	1.5176	1.5176	1.5175	**1.5152**
	Twitter	11.5381	11.3976	11.4932	11.3423	**11.1336**
		1.5170	1.5167	1.5169	1.5168	**1.5066**
	LiveJournal	56.2223	55.5147	54.9321	54.5475	**51.3682**
		5.1778	5.1234	5.0935	4.9499	**4.3945**
	Orkut	127.3402	121.4700	121.3921	121.7472	**113.4694**
		10.2619	9.6172	9.5319	9.5026	**8.7127**

Acknowledgments. This study was supported by the National Key Research and Development Program of China (Grant No. 2024YFB3310202).

References

1. Malewicz, G., Austern, M. H., Bik, A. J. C., et al.: Pregel: a system for large-scale graph processing. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, pp. 135–146 (2010)
2. Gonzalez, J. E., Xin, R. S., Dave, A., et al.: GraphX: Graph Processing in a Distributed Dataflow Framework. In: 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14), pp. 599–613 (2014)
3. Gonzalez, J. E., Low, Y., Gu, H., Bickson, D., Guestrin, C.: PowerGraph: Distributed graph-parallel computation on natural graphs. In: Symposium on Operating Systems Design and Implementation, pp. 17–30 (2012)
4. Ayall, T. A., Liu, H., Zhou, C., et al.: Graph Computing Systems and Partitioning Techniques: A Survey. IEEE Access **10**, 118523–118550 (2022)
5. Andreev, K., Räcke, H.: Balanced graph partitioning. In: Proceedings of the Sixteenth Annual ACM Symposium on Parallelism in Algorithms and Architectures, pp. 120–124 (2004)
6. Bourse, F., Lelarge, M., Vojnovic, M.: Balanced graph edge partition. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1456–1465 (2014)
7. Leskovec, J., Kleinberg, J., Faloutsos, C.: Graph evolution: Densification and shrinking diameters. ACM Transactions on Knowledge Discovery from Data **1**(1), 2Ces (2007)

8. Xie, C., Yan, L., Li, W. J., et al.: Distributed power-law graph computing: Theoretical and empirical analysis. Advances in Neural Information Processing Systems **27** (2014)
9. Petroni, F., Querzoni, L., Daudjee, K., et al.: HDRF: Stream-based partitioning for power-law graphs. In: Proc. of the 24th ACM Int'l Conf. on Information and Knowledge Management, pp. 243–252 (2015)
10. Nicoara, D., Kamali, S., Daudjee, K., Chen, L.: Hermes: Dynamic partitioning for distributed social network graph databases. In: Proc. of the 18th Int'l Conf. on Extending Database Technology, pp. 25–36 (2015)
11. Mayer, R., Orujzade, K., Jacobsen, H.-A.: 2PS: High-Quality Edge Partitioning with Two-Phase Streaming. arXiv preprint arXiv:2001.07086 (2020)
12. Nishimura, J., Ugander, J.: Restreaming graph partitioning: Simple versatile algorithms for advanced balancing. In: Proc. of the 19th ACM SIGKDD Int'l Conf. on Knowledge Discovery and Data Mining, pp. 1106–1114 (2013)
13. Huang, J., Abadi, D. J.: Leopard: Lightweight edge-oriented partitioning and replication for dynamic graphs. Proceedings of the VLDB Endowment **9**(7), 540–551 (2016)
14. Mayer, C., Mayer, R., Tariq, M. A., et al.: Adwise: Adaptive window-based streaming edge partitioning for high-speed graph processing. In: Proc. of the 38th IEEE Int'l Conf. on Distributed Computing Systems, pp. 685–695 (2018)
15. Taimouri, M., Saadatfar, H.: RBSEP: A reassignment and buffer based streaming edge partitioning approach. Journal of Big Data **6**(1), 1–17 (2019)
16. Nicoara, D., Kamali, S., Daudjee, K., Chen, L.: Hermes: Dynamic partitioning for distributed social network graph databases. In: Proc. of the 18th Int'l Conf. on Extending Database Technology, pp. 25–36 (2015)
17. Mayer, C., Tariq, M. A., Mayer, R., et al.: Graph: Traffic-aware graph processing. IEEE Transactions on Parallel and Distributed Systems **29**(6), 1289–1302 (2018)
18. Li, H., Yuan, H., Huang, J., et al.: Group reassignment for dynamic edge partitioning. IEEE Transactions on Parallel and Distributed Systems **32**(10), 2477–2490 (2021)
19. Kong, D., Xie, X., Zhang, Z.: Clustering-based Partitioning for Large Web Graphs. In: 2022 IEEE 38th International Conference on Data Engineering (ICDE 2022), pp. 593–606 (2022)
20. Hollocou, A., Maudet, J., Bonald, T., et al.: A Streaming Algorithm for Graph Clustering. In: NIPS 2017 - Workshop on Advances in Modeling and Learning Interactions from Complex Data, pp. 1–12 (2017)
21. Newman, M. E. J.: Modularity and community structure in networks. Proceedings of the National Academy of Sciences **103**(23), 8577–8582 (2006)
22. McAuley, J., Leskovec, J.: Learning to discover social circles in ego networks. In: Proc. of the 25th Int'l Conf. on Neural Information Processing Systems, pp. 539–547 (2012)
23. Yang, J., Leskovec, J.: Defining and evaluating network communities based on ground-truth. In: Proc. of the ACM SIGKDD Workshop on Mining Data Semantics, pp. 1–8 (2012)
24. Li, H., Yuan, H., Huang, J., et al.: Edge Repartitioning via Structure-Aware Group Migration. IEEE Transactions on Computational Social Systems **9**(3), 751–760 (2022)