An Efficient Lossless Graph Summarization Method for Large Streaming Graphs

Qia Wang, Yi Wang *, Ying Wang
College of Computer and Information Science College of Software
Southwest University
ChongQing, China
1159838035@qq.com, *echowang@swu.edu.cn, waying95@swu.edu.cn

***Abstract*—Graph summarization aims to extract critical information from large graphs by creating summaries that represent the original data. Especially, real-world daily applications generate massive dynamic streaming graphs, representing as edge or node streams. How to efficiently generate compact and lossless graph summaries for large streaming graphs is still a challenging problem. This paper proposes an efficient and scalable lossless summarization method for streaming graphs, called Partition and Similarity-based Two-Stage Summarization (PSTSS). PSTSS uses a streaming graph partitioning algorithm as the first stage to generate coarse-grained summaries by on-the-fly subgraph partitioning. The second stage generates fine-grained supernodes within these coarse-grained supernodes through efficient node similarity calculation. Our experiments on six large datasets demonstrate that our method achieves better operational efficiency, compression rate, scalability, and graph query speed compared to state-of-the-art lossless summarization algorithms.**

***Index Terms*—graph summary; graph compression; complex network; streaming graph; graph partition**

I. Introduction

Graphs are applied to represent interconnected data for the applications including web graphs, social networks, communication networks, citation networks, and even protein-protein interactions. With the emergence of big data, the sizes of real-world graphs are growing at an unprecedented rate. For instance, as of June 30th, 2022, WeChat has reached a scale of billions with 1.299 billion monthly active accounts [1]. The World Wide Web's indexed web pages have surpassed at least 6.42 billion [2] by November 2022. Moreover, large graphs generated by the daily activities of real-world applications are normally dynamic graph streams, representing as edge streams or node streams. Therefore, efficient storage, querying and visualization of large streaming graphs is an urgent problem that needs to be solved to better support downstream applications.

Graph summarization is a crucial technique for representing large graphs in a concise manner, which is a trending topic in data mining. Most previous works have focused on static graphs [3]–[8]. These static graph summarization methods require importing the entire graph into memory for summarization which results in high memory requirements and poor scalability. Existing streaming graph summarization methods [9]–[13] generate lossy summaries and cannot achieve the accurate reconstruction of the original graph. Moreover, in incrementally summarization, they mainly consider the case of directly adjacent nodes. They lack recognition of the features of nodes in a more comprehensive local range, which affects the compression ratio and readability of the graph summary.

To address the shortcomings of the above graph summarization methods, this study proposes PSTSS (Partition and Similarity-based Two-Stage Summarization), an efficient lossless graph summarization method for large-scale streaming graphs. Our contributions are as follows:

- **Efficient Lossless Graph Summarization Algorithm** We propose PSTSS for lossless graph summarization. With linear scalability, PSTSS summarizes large-scale graph faster and achieves better compression than the state-of-the-art graph summarization method.
- **Query Evaluation Algorithm for PSTSS Summary** We propose neighborhood queries evaluation algorithm for PSTSS Summary. Neighborhood queries can be answered quickly from a summary graph and edge corrections.
- **Extensive Experiments** We confirmed that PSTSS outperforms 3 state-of-the-art graph summarization algorithms on 6 real graphs in different domains.

II. Related Work

Graph summarization is a known NP-hard problem [14]. Previous research has extensively explored state-of-the-art methods in graph summarization, including a comprehensive review of existing algorithms [15], [16]. This section specifically discusses lossless and lossy summarization techniques as well as streaming graph summarization, which are related to our study closely.

A. Lossless Summarization

The goal of lossless summarization is to identify the most concise summary graph $G^* = (S, P)$ that can accurately recreate the initial graph. Navlakha et al. [3] were the pioneers in solving the problem of lossless graph summarization using the minimum description length (MDL) principle [17]. They introduced two heuristics, Randomized and Greedy, to

This research is sponsored by the Educational Reform Research Project of Southwest University (2022JY085) and the Fundamental Research Funds for the Central Universities-Doctoral Fund (SWU222001).
DOI reference number: 10.18293/SEKE2023-107

minimize the summary graph and edge corrections through random selection and greedy strategies respectively. Khan et al. [4] employed unified locality sensitive hash (ULSH) to quickly select node pairs for merging, requiring a relatively high computational cost. Shin et al. [5] grouped node pairs using min-hash before merging highly similar nodes in each group. Ko et al. [9] on the other hand, maintain a streaming graph summary incrementally in response to edge additions and deletions while ensuring compression ratio and efficiency.

B. Lossy Summarization

The goal of lossy graph summarization is to create a brief summary graph that preserves the neighbors of each node in the original decompressed graph. Two algorithms, APXMDL [3] and SWEG-lossy [5], are used for this purpose. Both algorithms take edge corrections into account in their output results to minimize the most concise representation $G^* = (S, P, C^+, C^-)$ of the original graph. Other algorithms, such as K-GS [6], S2L [7], and SSUMM [8], represents the summary graph as $G^* = (S, P)$ without considering edge corrections. The K-GS algorithm selects node pairs from a pool of candidate nodes and merges them repeatedly to decrease the adjacency matrix of both the input graph and its reconstructed output representation. Meanwhile, S2L guarantees an approximate "p-reconstruction error" for the input graph by utilizing geometric clustering to summarize it. Finally, SSUMM applies the MDL principle to balance size and accuracy of summarising by sparsifying the original graph for lossy summarization purposes.

C. Streaming graph Summarization

Streaming graph summarization is more challenging than static graph summarization due to the dynamic of streaming data. Algorithms of streaming graph summarization mainly use statistical methods to process and analyze basic structures in the streaming graph, such as estimating edge frequency and node degree distribution. For example, MoSSo groups similar nodes and incrementally calculates a lossless summary, while gSketch [19] estimates edge frequency to generate a lossy summary that supports structural queries. GS4 [10] generates a lossy summary using the sliding window model and vertex properties of the graph stream. In literature [11], a compressed binary tree corresponds to the streaming graph data for lossy summarization. In literature [12], hash functions maintain a minimum neighborhood sample subgraph in real-time. GSS [13] first generates a sketch of the streaming graph using hash functions, then uses a novel data structure to store it, achieving lossy summarization supporting various queries.

Mainly of the streaming graph summarization algorithms mentioned above are lossy and focus primarily on directly adjacent nodes. However, they fail to recognize the features of nodes within a broader local range, which can impact both compression ratio and readability of the graph summary.

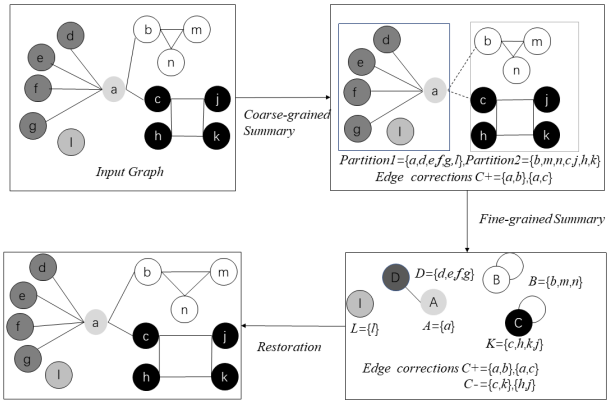

Fig. 1. Partition and Similarity-based Two-Stage Summarization (PSTSS)

III. Two-stage lossless summarization method for streaming graph

A. Overview of PSTSS

PSTSS adopts a divide-and-conquer algorithm to partition the streaming graph online and generate coarse-grained summary by dividing it into multiple subgraphs. Additionally, fine-grained supernodes are generated through efficient node similarity calculation. Fig. 1 illustrates the two-stage summarization process. Firstly, a greedy algorithm partitions the streaming graph [20], [21] to generate a coarse-grained summary of the graph. Secondly, each coarse-grained supernode is summarized separately using a node similarity algorithm to obtain fine-grained supernodes along with their corresponding sets of superedges and edge corrections. This lossless summarization method enables reconstruction of the original graph.

B. Streaming Graph Summary Problem Description

This study focuses on streaming (undirected) graphs represented as node streams, which consist of a sequence of nodes with their corresponding neighbor lists. Alternatively, streaming graphs can also be represented as edge streams, but for the purposes of this study, they will not be considered. Given a node streaming graph $G_t = (v_t, N(v)_t)_{t=0}^{t=\infty}$,where v_t is a node and $N(v_t)$ is the set of neighboring nodes of v_t . $G = (V, E)$ is a snapshot of the streaming graph at a given moment, and V and E are the set of nodes and the set of edges of the streaming graph, respectively. The lossless summary of G is $G^* = (S, P)$ and the set of edge corrections C, where S is the set of supernodes, P is the set of supernodes, and C is the set of edge corrections $C = (C^+, C^-)$.

For a given streaming graph G, the goal of lossless graph summarization is to efficiently generate a compact summary that can be used to reconstruct the original graph without losing any nodes or edges.

C. Coarse-grained Summary Method Based on Greedy Partitioning Algorithm

This study uses a streaming graph partitioning algorithm to partition the streaming graph and create a coarse-grained

TABLE I
TABLE OF SYMBOLS

Symbol	Meaning
$N(v)$	neighborhood of node v
$C = (C^+, C^-)$	edge corrections
C^+	set of edges to be inserted
C^-	set of edges to be deleted
k	number of partitions, $k \in N$
Pa_i	a set of vertices in a partition i, $i \in [1, k]$
$G_i = (Pa_i, E_i)$	subgraph after graph partition
$\pi = \{G_1, G_2..., G_k\}$	a set of subgraphs(Coarse-grained Summary)
$S(v)$	supernode in G^* that contains node v
S	a set of supernodes in G^*
$G^* = (S, P)$	a summary graph with supernodes S and superedges P

summary. The algorithm takes a sequence of nodes with their neighbor lists as input (node stream) and partitions them into subgraphs based on load balancing principles while minimizing edge cuts. Unlike traditional graph partition algorithms, this process only requires one graph traversal, making it suitable for large-scale graphs with lower time complexity. References include [22], [23]. For an undirected graph $G = (V, E)$, the node division of the graph divides the nodes in the graph G into k partitions, each of which is denoted by $Pa_i (i \in \{1, ..., k\})$ and satisfies the following three conditions: 1) $Pa_i \neq Pa_j (i, j \in \{1, ..., k\}, i \neq j)$; 2) $\cup Pa_i = V$; 3)$\emptyset \neq Pa_i \subseteq V$.

Definition 1 Coarse-grained summary of streaming graph Given a node streaming graph $G_t = (v_t, N(v)_t)_{t=0}^{t=\infty}$, $\pi = \{G_1, G_2..., G_k\}$, where $G_i = (Pa_i, E_i)(i \in \{1, ..., k\})$is the subgraph after division, and $eC \subseteq E$ is the set of edge cuts resulting from the division of the set of nodes. π and eC are the coarse-grained summary of the streaming graph.

Algorithm 1 is given the number of divisions k. Firstly, the original graph is read in real-time as a sequence of nodes and their neighborhood; Secondly, under the premise of satisfying load balancing, the nodes are divided according to equations (1) and (2), and the read nodes are divided into partitions that maximize the value of Equation (1); Finally, the coarse-grained summary is calculated, and the edges connecting each subgraph are saved as edge corrections.

$$g(v, Pa_i) = |Pa_i \cap N(v)|(1 - \frac{|Pa_i|}{C_a}) \tag{1}$$

$$f(v) = argmax_{i \in [1,k]} \{g(v, Pa_i)\} \tag{2}$$

In Equation (1), the partition capacity $Ca = |V|/k$, node division needs to ensure the load balancing degree to achieve the division of the nodes of the balanced graph as much as possible. $g(v, Pa_i)$ denotes the value of load balancing degree with $|Pa_i \epsilon N(v)|$, and $f(v)$ in Equation (2) denotes the division of node v into partitions that maximize the value of $g(v, Pa_i)$.

Algorithm 1 computes a coarse-grained summary of the resulting streaming graph based on the greedy partitioning algorithm described above. Let the number of partitions be k. Algorithm 1 traverse every node in the graph, so the space complexity is $O(V)$ and the time complexity is $O(k|V|+|E|)$, where V and E are the set of nodes and the set of edges of the graph at the current moment.

Algorithm 1 Coarse-grained summary of streaming graph

Input: $G_t = \{v_t, N_t(v)\}$
Output: G_i and eC

```
for i = 1 to i = k do
    w_i = 1 − |Pa_i|/C_a
    g(v, Pa_i) = |Pa_i ∩ N(v)| ∗ w(i)
end for
for i = 1 to i = k do
    ptno = argmax(g(v, Pa_i))
end for
Pa_ptno ← Pa_ptno ∪ v_t
C.add(E_t)
Return G_i and eC
```

Example 1 Be provided with an undirected graph G shown in Fig. 1, containing 13 nodes and 11 edges, and determine the number of partitions k = 2. Under the premise of satisfying the capacity constraint $Capacity = 11/2 = 6$, $g(v, Pa_i)$ is computed separately for all possible partitioning cases into two subgraphs, then the maximum value of $g(v, Pa_i)$ is selected among all partitioning results, which is the two subgraphs G_1 and G_1 for graph partitioning. The process of graph partitioning generates edge cut sets, i.e., edge $\{a, b\}$ and edge $\{a, c\}$, which are saved as edge cut set eC supports the reconstruction of the graph.

D. Fine-grained Summary Method Based on Node Similarity

This study uses coarse-grained summarization to perform fine-grained supernode calculation. The method is based on node similarity, which groups nodes by identifying their structural similarities. This allows a supernode to replace a group of nodes in the resulting fine-grained summary.

Definition 2 Fine-grained summary of streaming graph Given a node streaming graph $G_t = (v_t, N(v)_t)_{t=0}^{t=\infty}$, the coarse-grained summary is obtained as π and eC. And for each subgraph, $G_i = (Pa_i, E_i)(i \in \{1, ..., k\})$ in the coarse-grained summary, the nodes are merged according to the node similarity to form the fine-grained supernodes. The superedges and edge corrections are generated to obtain the fine-grained summary of the streaming graph $G^* = (S, P)$ and edge corrections C.

Algorithm 2 describes the fine-grained summary method. Algorithm 2 is divided into three main tasks: 1) calculating node pairs to compute similarity and sorting node pairs by similarity; 2) merging highly similar nodes; 3) connecting supernodes to form super edges and edge correction.

Selection of Fine-grained Supernode Candidates When dealing with large graphs that have node bases of orders of magnitude, evaluating all possible pairs of directly or indirectly connected nodes to determine supernodes is inefficient. To address this issue, PSTSS introduces the locality-sensitive

hashing (LSH) method which accurately identifies similar nodes. LSH calculates similarity by comparing hash codes of each node's neighborhood. Nodes falling into different buckets are considered dissimilar and their similarity is set to 0; these node pairs are not merged. This approach limits the computation of node similarity to only those in the same bucket, significantly reducing computational resources and running time while still identifying similar nodes.

Algorithm 2 Fine-grained summary of streaming graph

Input: $\pi, eC, h(HashFunction), \tau(SimilarityThreshold)$
Output: $G^* = (S, P), C$

for $G_i \in \pi$ **do**
  for $v \in G_i$ **do**
    Create minhash signature column, MSC_v, using neighbors of v by h hash functions
  end for
  for $G_i \in \pi$ **do**
    compute $Sim(v, u)$ for each bucket
    $Sim(v, u) \leftarrow 0$ for v and u in different buckets
    Rank AdaSim scores to generate F
    Pop v, u from F
    if $Sim(v, u) > \tau$ **then**
      merge v, u and update S
    end if
  end for
end for
for $A \in S$ **do**
  for $B \in S$ **do**
    if $E_{AB} \neq \emptyset$ and $|E_{AB}| \leq (\frac{|T_{AB}|+1}{2})$ **then**
      $C^+ \leftarrow C^+ \cup E_{AB}$
    else
      $P = P/cupA, B$
      $C^- \leftarrow C^- \cup (T_{AB} - E_{AB})$
    end if
  end for
end for

Method of calculating node similarity Algorithm 2 calculates the similarity of node pairs using the recursive similarity measure in AdaSim [24]. In some of the existing similarity calculation methods, only the common neighboring nodes directly connected to the selected node pair are often considered; In this method, indirectly connected nodes are also considered. Equations (3) and (4) give the formulas for calculating the iterative form of AdaSim. When $l = 0$, $Sim_0(a, b) = 1$ if $a = b$, and $Sim_0(a, b) = 0$ if $a \neq b$. $D \in (0, 1)$ is the determining damping factor, the parameter $\theta \in (0, 1]$ is the important parameter, and m is the maximum value of Ada similarity. First, the weights of the common neighbor nodes directly connected to nodes a and b (with path length 1) are summed and calculated to obtain the similarity $Sim_1(a, b)$ in the first step. Next, the common nodes indirectly connected with nodes a and b at a distance of path length two are iteratively computed. To balance the computational efficiency and similarity accuracy, this study considers the node similarity within two steps, i.e., $Sim_2(a, b)$ is used as the similarity of nodes.

$$Sim_l(a,b) = D * (\frac{\theta}{m} \sum_{i \in N(a) \cap N(b)} wt_i + \frac{1-\theta}{\sum_{r \in N(a)} wt_r \sum_{t \in N(b) wt_t}} \sum_{i \in N(a)} \sum_{j \in N(b)} wt_i * Sim_{l-1}(i,j) * wt_j \tag{3}$$

$$wt_i = \frac{1}{log(|N(i)| + e)} \tag{4}$$

After calculating the similarity of node pairs, the node pairs with high similarity need to be merged. The node pair sequence F is obtained by fast sorting according to the likeness of node pairs, setting the similarity threshold τ, and selecting node pairs greater than τ from the node pair sequence for merging according to the similarity. After generating the supernodes, the method connects the supernodes to form superedges and edge corrections according to the best encoding.

Optimal encoding For each supernode pair $\{u, v\}$, let $E_{AB} = \{\{u, v\} \in E | u \in A, v \in B, u \neq v\}$ and $T_{AB} = \{\{u, v\} \subseteq V | u \in A, v \in B, u \neq v\}$ be the set of existing and potential edges between A and B, respectively. The edges between A and B (i.e., E_{AB}) are encoded according to the following rules.

(1) If $|E_{AB}| \leq (|T_{AB} + 1|)/2$, then add all edges in E_{AB} to C^+;

(2) If $|E_{AB}| > (|T_{AB}+1|)/2$, then add the superedge A, B to P and $T_{AB} \setminus E_{AB}$ to C^-.

Complexity Analysis Algorithm 2 is divided into three main tasks. The time complexity of task 1 is $O(E)$, and the space complexity is $O(V)$, where E is the base of the set of edges in the original graph and V is the base of the set of nodes in the original graph; Task 2 first applies the LSH algorithm in the graph with time complexity of $O(E)$, and then calculates the similarity and sorts the node pairs for each bucket with space complexity of $O(V)$. The time complexity of sorting the sequence of node-pairs is $O(FlogF)$, where F is the number of node pairs; The time complexity of task 3 is $O(E)$, and the space complexity is $O(V)$. In summary, the space complexity of Algorithm 2 is $O(V)$, and the time complexity is $O(E + FlogF)$.

Example 2 The undirected graph G shown in Fig. 1, is divided to generate a coarse-grained summary, and then two subgraphs, G_1, G_2, and edge correction C are formed. Then the nodes are fine-grained summarized according to similarity. Firstly, the LSH algorithm is used to group the nodes and calculate the AdaSim similarity, merge the highly similar nodes to form the supernodes and connect the superedges, and finally obtain the summary graph $G^* = (S, P)$ and edge correction C. Note that in the process of grouping the nodes to form the supernodes, unlike calculating the similarity within only one step, this study considers the nodes at a

distance of two steps to calculate the parallel, which more accurately captures the local similarity features of the nodes. For example, nodes c, j, h, k are merged into one supernode based on the high similarity of AdaSim within two steps.

E. Query Evaluation Algorithm for Summary Graph

Neighborhood queries are a key building block that is reused in many graph algorithms (Dijkstra algorithm, PageRank, ShorestPaths, etc.).Algorithm 3 describes how to answer neighbor queries (given a node $v \in V$, find the neighbors $N(v)$ of v) on a summary graph $G^* = (S, P)$ and edge correction$C = (C^+, C^-)$ without reconstructing the original graph. Let the neighbors of node v in edge corrections C^+ be $N^+(v)$ and the neighbors of node v in edge corrections C^- be $N^-(v)$.

Algorithm 3 Neighbor Query Processing on PSTSS summary

Input: summary graph $G^* = (S, P)$, edge corrections $C = (C^+, C^-)$, query node $v \in V$
Output: $N(v)$

```
if S(v) has a self-loop in G then
    N(v) = N(v) ∪ (S(v) − {v})
end if
if for each neighbor A(≠ S(v)) of S(v) in G* then
    N(v) = N(v) ∪ A
end if
N(v) = (N(v) ∪ N+(v)) − N−(v)
Return N(v)
```

IV. EXPERIMENTS

A. Experimental Settings

Experimental environment:Intel Core i5 and 8 GB main memory, having 64 bit Windows 10 Professional edition. **Comparison algorithms**:Experimentally, three state-of-the-art algorithms are selected from the lossless graph summarization algorithms as comparison algorithms. (1) SAGS [4] (h=30, b=10, p=0.3) (2) SWeG [5] (T=10,e =0) (3) MoSSo [9] (e = 0.3, c = 120).

B. Datasets

To verify the effectiveness of PSTSS, six large graph datasets from different domains (Table II) were selected for the experiments of running efficiency and compression rate comparison.In every graph in Table II, the experiment ignores the direction of all edges and removed both self-loops and multiple edges.

C. Speed

The experiment first compared the running time of the PSTSS algorithm with the lossless graph summarization algorithms SAGS, SWeG, and MoSSo. The specific running time comparison results are shown in Table III. With the same datasets selected for graph summarization, the PSTSS algorithm proposed in this paper has higher summarization efficiency than the other three lossless graph summarization comparison algorithms on all six datasets. Among them, the PSTSS algorithm significantly outperforms the other lossless summarization methods on the dataset Ego-Facebook.

TABLE II
DATASETS

Name	Nodes	Edges	Summary
Protein	6,229	146,160	Protein Interaction
UK-2007	100,000	3,216,152	Hyperlinks
CNR-2000	325,557	5,565380	Hyperlinks
LiveJournal	4,847,571	68,993,773	Social
Ego-Facebook	4,039	88,234	Social
Email-Enron	36,692	183,831	Email

TABLE III
RUNNING TIME(SECONDS)

Dataset	SAGS	SWeG	MoSSo	PSTSS
Protein	$1.84 * 10^2$	$1.44 * 10^2$	$1.24 * 10^2$	$0.91 * 10^2$
UK-2007	$1.89 * 10^4$	$1.39 * 10^4$	$1.28 * 10^4$	$0.86 * 10^4$
CNR-2000	$1.24 * 10^5$	$1.87 * 10^5$	$1.45 * 10^5$	$0.97 * 10^5$
LiveJournal	$1.66 * 10^5$	$1.50 * 10^5$	$1.57 * 10^5$	$1.88 * 10^5$
Ego-Facebook	$1.29 * 10^1$	$1.24 * 10^1$	$1.17 * 10^1$	$0.69 * 10^1$
Email-Enron	$1.21 * 10^3$	$0.91 * 10^3$	$1.14 * 10^3$	$0.79 * 10^3$

D. Compression Ratio

To verify the effectiveness of the PSTSS algorithm, the experimental uses the compression ratio defined in Equation (5) to measure the relative size of the summary graph to the original graph. With the same original graph, the smaller the compression ratio value, the more compact the output graph is proved to be, and the more influential the algorithm is in compressing the original graph.The experiments were performed to summary the datasets in Table II by three comparison algorithms and PSTSS, and the specific experimental results are shown in Fig. 2.

$$RN = |S|/|V| \tag{5}$$

As shown in Fig. 2, the compression ratio of the proposed summarization method PSTSS algorithm in this study outperforms all three comparison algorithms, SAGS, SWeG, and MoSSo.

E. Scalability

To measure the scalability of the PSTSS algorithm, experiments were designed to test the variation of the running time of the PSTSS algorithm on datasets of different sizes. Experiment sampled different numbers of nodes from the LiveJournal dataset and generated multiple graph datasets and used them as input graphs. As shown in Fig. 3, the running time of the PSTSS algorithm increases linearly with the number of nodes in the input graph with a slope close to 1, which indicates that the PSTSS algorithm has great scalability.

F. Neighbor quries on query PSTSS summary

The summary results generated by the PSTSS algorithm can significantly reduce the query time to the original graph. To verify this property of the PSTSS algorithm, experiments on neighbor node queries are designed in this study. The experiments were performed using the results of the PSTSS

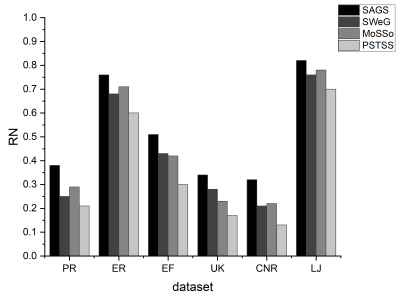

Fig. 2. Relative size of outputs

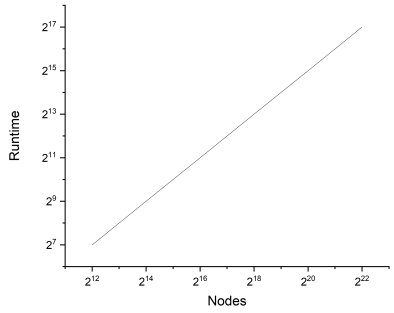

Fig. 3. scalability

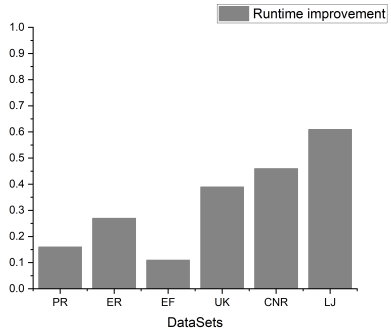

Fig. 4. Runtime improvement

algorithm in the dataset of Table II for neighbor queries, running Algorithm 3 and comparing the queries times with the SWeG algorithm. Fig. 4 shows the reduction in running time.

V. Conclusion

This study proposes a two-level lossless summarization method called PSTSS to address the challenge of large streaming graph summarization. The method is based on streaming graph partitioning and node similarity. Results from experiments conducted on six large graph datasets demonstrate that compared to three typical lossless graph summarization algorithms, PSTSS has lower time complexity, better compression rate, higher operational efficiency, good scalability, and can improve the query efficiency of graphs. It can effectively implement large graph summarization.

Future research directions include exploring parallel streaming graph summarization and combining stream graph summarization methods with techniques such as graph visualization and mining.

References

[1] Tencent Tencent Announces Second Quarter and Interim Results for 2022. ([EB/OL],2022), https://static.www.tencent.com/uploads/2022/08/17.pdf.

[2] WorldWideWeb The size of the World Wide Web (The Internet). ([EB/OL],2022), https://www.worldwidewebsize.com.

[3] Navlakha, S., Rastogi, R. & Shrivastava, N. Graph summarization with bounded error. *Proceedings Of The 2008 ACM SIGMOD International Conference On Management Of Data*. pp. 419-432 (2008).

[4] Khan, K., Nawaz, W. & Lee, Y. Set-based approximate approach for lossless graph summarization. *Computing*. **97** pp. 1185-1207 (2015).

[5] Shin, K., Ghoting, A., Kim, M. & Raghavan, H. Sweg: Lossless and lossy summarization of web-scale graphs. *The World Wide Web Conference*. pp. 1679-1690 (2019).

[6] LeFevre, K.& Terzi, E. GraSS: Graph structure summarization. *Proceedings Of The 2010 SIAM International Conference On Data Mining*. pp. 454-465 (2010).

[7] Riondato, M., Garcıa-Soriano, D.& Bonchi, F. Graph summarization with quality guarantees. *Data Mining And Knowledge Discovery*. **31** pp. 314-349 (2017).

[8] Lee, K., Jo, H., Ko, J., Lim, S. & Shin, K. Ssumm: Sparse summarization of massive graphs. *Proceedings Of The 26th ACM SIGKDD International Conference On Knowledge Discovery Data Mining*. pp. 144-154 (2020).

[9] Ko, J., Kook, Y. & Shin, K. Incremental lossless graph summarization. *Proceedings Of The 26th ACM SIGKDD International Conference On Knowledge Discovery Data Mining*. pp. 317-327 (2020).

[10] Ashrafi-Payaman, N., Kangavari, M., Hosseini, S. & Fander, A. GS4: Graph stream summarization based on both the structure and semantics. *The Journal Of Supercomputing*. **77** pp. 2713-2733 (2021).

[11] Nelson, M., Radhakrishnan, S. & Sekharan, C. Queryable compression on time-evolving social networks with streaming. *2018 IEEE International Conference On Big Data (Big Data)*. pp. 146-151 (2018).

[12] Bandyopadhyay, B., Fuhry, D., Chakrabarti, A. & Parthasarathy, S. Topological graph sketching for incremental and scalable analytics. *Proceedings Of The 25th ACM International On Conference On Information And Knowledge Management*. pp. 1231-1240 (2016).

[13] Gou, X., Zou, L., Zhao, C. & Yang, T. Graph stream sketch: Summarizing graph streams with high speed and accuracy. *IEEE Transactions On Knowledge And Data Engineering*. (2022).

[14] Liu, X., Tian, Y., He, Q., Lee, W. & McPherson, J. Distributed graph summarization. *Proceedings Of The 23rd ACM International Conference On Conference On Information And Knowledge Management*. pp. 799-808 (2014).

[15] Liu, Y., Safavi, T., Dighe, A. & Koutra, D. Graph summarization methods and applications: A survey. *ACM Computing Surveys (CSUR)*. **51**, 1-34 (2018).

[16] Wang Xiong, Dong Yihong, Shi Weijie Pan Jianfei. Progress and Challenges of Graph Summarization Techniques[J]. Journal of Computer Research and Development, 2019, 56(6): 1338-1355.

[17] Barron, A., Rissanen, J. & Yu, B. The minimum description length principle in coding and modeling. *IEEE Transactions On Information Theory*. **44**, 2743-2760 (1998).

[18] Durbeck, L. & Athanas, P. DPGS Graph Summarization Preserves Community Structure. *2021 IEEE High Performance Extreme Computing Conference (HPEC)*. pp. 1-9 (2021).

[19] Zhao, P., Aggarwal, C. & Wang, M. gsketch: On query estimation in graph streams. *ArXiv Preprint ArXiv:1111.7167*. (2011).

[20] Stanton, I. Streaming balanced graph partitioning algorithms for random graphs. *Proceedings Of The Twenty-fifth Annual ACM-SIAM Symposium On Discrete Algorithms*. pp. 1287-1301 (2014).

[21] Stanton, I. & Kliot, G. Streaming graph partitioning for large distributed graphs. *Proceedings Of The 18th ACM SIGKDD International Conference On Knowledge Discovery And Data Mining*. pp. 1222-1230 (2012).

[22] Wang, X., Chen, W., Yang, Y. & Others Research on Knowledge Graph Partitioning Algorithms: A Survey. *Chinese Journal Of Computers*. **44**, 235-260 (2021)

[23] Abbas, Z., Kalavri, V., Carbone, P. & Vlassov, V. Streaming graph partitioning: an experimental study. *Proceedings Of The VLDB Endowment*. **11**, 1590-1603 (2018).

[24] Rehyani Hamedani, M. & Kim, S. AdaSim: A Recursive Similarity Measure in Graphs. *Proceedings Of The 30th ACM International Conference On Information Knowledge Management*. pp. 1528-1537 (2021).