2024 5th International Conference on Electronic Communication and Artificial Intelligence (ICECAI) | 979-8-3503-8694-3/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICECAI62591.2024.10675249

$\mathcal{PR}_B$-Sketch: A Graph Stream Summarization Model Based on Pattern and Rank

Xinyao Chen
School of Computer and Communication Engineering,
Northeastern University at Qinhuangdao, Qinhuangdao, China
13643425080@163.com

Xiujing Wen
School of Computer and Communication Engineering,
Northeastern University at Qinhuangdao, Qinhuangdao, China
18340868846@163.com

Changyong Yu*
School of Computer and Communication Engineering,
Northeastern University at Qinhuangdao, Qinhuangdao, China
* Corresponding author: cyyneu@126.com

***Abstract*—Graph stream means that the graph's edges are sequentially updated as a stream, which has essential applications in network security and social networks. Different from static graphs, the graph stream has the characteristics of massiveness and fast dynamic update speed. How to efficiently summarize graph streams has become a key research topic. Given a graph, graph data stream summarization is to design a sketch that costs a much smaller space. However, the existing graph stream summarization algorithms have defects, such as only supporting limited query types and low hash space utilization. In this article, we propose a new data structure: a graph data stream summarization method based on Pattern and Rank (referred to as $\mathcal{PR}_B$-Sketch) that is a storage structure with high accuracy and high hash space utilization. $\mathcal{PR}_B$-Sketch can support multiple types of graph query algorithms. Both theoretical analysis and experimental results show that our method is superior to the prior arts in terms of query accuracy and hash space utilization when processing massive graph stream data.**

Keywords- Graph stream, Pattern, Rank, Graph query, Hash space utilization

I. INTRODUCTION

Nowadays, the data stream technique plays an important role, and new challenges are facing existing technologies in the era of big data. Due to the huge size and fast update speed, the traditional data summarization algorithms cannot directly and effectively be used to store the graph data stream and support a variety of graph queries. Therefore, the problem of graph stream summarization has become a research hotspot and attracted the attention of many scientific researchers in the database fields.

There are several methods for summarizing data streams, and we divided them into two categories according to whether they can process graph streams. One category comprises algorithms that cannot directly handle graph streams, such as CM [1]. The other category includes algorithms explicitly tailored for the processing of graph stream data, including TCM [2], gMatrix [3], LGS [4], and GSS [5], as well as other specialized methods. Recently, many new methods have been proposed, such as Dolha [6], GS4 [7], Scube [8], Cuckoo Matrix [9], and many more advanced graph sketch techniques [10, 11]. These algorithms attempt to solve the problem of summarizing the huge-size graph stream on different aspects, and they have made progress. However, most of the methods are implemented based on the hash method. The inevitable hash collision causes the low utilization of the hash space, especially when dealing with graph stream data. This problem leads to the stability and accuracy of these methods are sometimes low. Focusing on the above issues, we propose $\mathcal{PR}_B$-Sketch, an efficient model for graph stream summarization. The key contributions of this paper are as follows:

1) We proposed the Pattern and Rank methods in basic $\mathcal{PR}_B$-Sketch to improve the hash space utilization, which enables each edge to utilize multiple hash units.

2) We performed experiments on real-world data sets, and the results show that our method is better than the state-of-the-art methods in terms of hash space utilization and query accuracy when processing massive graph stream data.

II. PROBLEM DEFINITIONS

Table 1 shows the symbols in the article, with their definitions and explanations.

Definition 1 (Graph Stream). A graph stream is a continuous sequence of data items without boundary, denoted by $S = \{e_1, e_2, \dots e_n\}$ each item $e_i = ((s, d); t; w)$ represents a directed edge at time t, where the source node is s and the destination node is d. Therefore, the sequence S changes with each item e_i to form a dynamic directed graph $G = (V, E)$, where V and E represent the node set and edge set of the graph respectively. We call G a stream graph. The edge (s, d) may appear at points of different time in the graph stream data. Its weight is the sum of the weights of the edges formed by the same source node and destination node. In this article, we default that the weight of each edge is greater than 0.

Definition 2 (Graph Sketch). A sketch of graph $G = (V, E)$ is a smaller graph $G_h = (V_h, E_h)$, using a hash function $H(*)$ to map the node set to V_h and the edge set to E_h, where $|V_h| \leq |V|$, $|E_h| \leq |E|$.

Table 1. Symbols

Symbols	Definition and explanation
s/d	Source node/destination node
G	Stream Graph
V/E	Node set/edge set
$H(*)$	Hash value of *

Γ	An Aggregation function, which can be a summation function, a minimum function or an average function, etc.
L	Optional parameters
vR	Two-dimensional array for storing random arrangement
W	Set of hash values
D	Conflict upper limit of the conflict list
w_t	The weight of the new edge at a certain moment
$[M]$	Integer not exceeding N
$\Pi(L)$	the set of permutations of elements in set $\{1,2,\dots,L-1\}$
$suffix(U,l)$	the l length suffix of vector U

Definition 3 (Graph Data Stream Summarization). The problem of graph stream summarization is to design a graph sketch $G_h = (V_h, E_h)$, and the data structure corresponding to the graph sketch.

Definition 4 (Edge query). Estimate the overall frequency of edge (x, y), $\tilde{f}(x,y) = \sum_{t_i \in T} f(x,y,t_i)$.

Definition 5 (Reachability query). Estimate whether there is a reachable path between two nodes x and y.

III. BASIC $\mathcal{PR}_B$-SKETCH

A. Model definition

Definition 6 (Pair-wise independent hash functions). Used to reduce hash conflicts and improve query accuracy. $\mathcal{H} = \{h: [N] \to [M]\}$ represents a pair of independent hash function family [12, 13], if $i \neq j \in N$, and $k, l \in M$, $Pr_{h \leftarrow \mathcal{H}}[h(i) = k \wedge h(j)=l] = 1/M^2]$ holds.

Definition 7 ($\boldsymbol{\mathcal{PR}_B}$-Sketch). $\mathcal{PR}_B$-Sketch is a four-tuple, represented by $\{gM, gP, gR, gH_v\}$, where gM is a three-dimensional array, $gM[i,j,z] = (rank, c, list)$ represents the hash cell in the i^{th} row, j^{th} column and z^{th} layer, and $rank$ represents the rank value of the edge, c represents the weight after the update, $list$ is a hash conflict list; $gH_v[i]: V \to W, i = 1, 2 \dots v$ represents v paired independent hash functions; $gP: E \to (W^2)^{L^2}$ is a vector-valued function; $gR: E \to \Pi(L^2)$ is a map from edge to rank vector.

The basic structure of $\mathcal{PR}_B$-Sketch is shown in part (B) of Figure 1. It is a three-dimensional structure of $w \times w \times v$. Each hash function maps the node in set V to an integer in the range $[0, w-1]$. The shaded grid coordinates of part B in Figure 1 is $(h_1(s), h_1(d), 1)$.

The pattern construction method based on double hash is as follows:

$$P(s)[j] = \left(x_j\, H_v[m](s) + y_j H_v'[m](s)\right) mod\ w \quad (1)$$

Where $x_j, y_j, j = 0, \dots, L-1$ is the given L pairs of parameters, $H_v[m](*)$ is the m^{th} hash function in $\mathcal{PR}_B$-Sketch, $H_v'[m](*)$ is any hash function independent of $H_v[m](*)$.

In the $\mathcal{PR}_B$-Sketch, we use the above method of constructing pattern to construct the pattern of the source node s and the destination node d respectively. Then the function $gP: E \to (W^2)^{L^2}$ can be defined as follows:

$$gP(s,d)[i \times L + j] = (P(s)[i], P(d)[j]) \quad (2)$$

Where $(P(s)[i], P(d)[j])$ represents the element of Cartesian product of set $P(s)$ and $P(d)$. As shown in Figure 1, the pattern of the source node s of the incoming edge e_t is $P(s)$, and the pattern of the destination node d is $P(d)$. Then the pattern $P(e_t)$ of the edge e_t obtained by the Cartesian product is shown in the part (D) of Figure 1.

Given an integer L, we first generate $\Pi(L)$ and store it into a file with one permutation in a line without ordering. We use vR to the represent the vector of permutations of $\Pi(L)$. The rank function R in the m^{th} row of the hash table is shown as follows:

$$R(s)[j] = \begin{cases} 1 & j = L-1 \\ vR\left[\left(H_v[m](s)\right) mod\ |\Pi(L)|\right][j] + 1 & j < L-1 \end{cases}$$

The rank of node s, $R(s)$, is a permutation of elements in $\{1,2,\dots,L\}$, in which 1 is the highest level and L is the lowest level.

To obtain the rank of the edge (s,d), similarly we use the above method of constructing rank to construct the rank vectors of the source node s and the destination node d respectively. Then the function $gR: E \to \Pi(L^2)$ can be defined as follows:

$$gR(s,d)[i \times L + j] = L \times (R(s)[i] - 1) + R(d)[j] \quad (3)$$

Where $i, j = 0,1,\dots,L-1$ represents the index of rank vector $R(s)$ and $R(d)$. As shown in part (C) of Figure 1, the rank of the source node s of the incoming edge e_t is $R(s)$, and the rank of the destination node d is $R(d)$. Then the rank $gR(e_t)$ of the edge e_t obtained by the above formula for constructing rank is shown in part (D) of Figure 1.

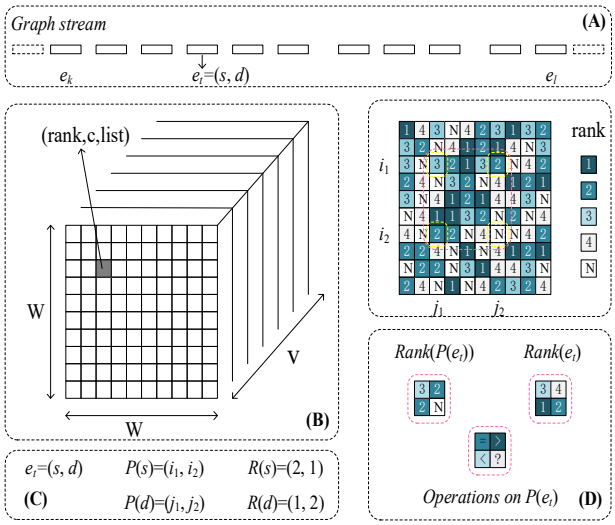


Figure 1. The basic structure and processing of $\mathcal{PR}_B$-Sketch.

B. Model update

The update process of $\mathcal{PR}_B$-Sketch is shown in Algorithm 1. As shown in Algorithm 1, let p and q indicates $gR(s,d)[j]$ and $gM[gP(s,d)[j]][z].rank$ respectively, there are three cases:

- Evict and Occupy: In the case of $p < q$, then the value $gM[gP(s,d)[j]][z].rank$ is changed to $gR(s,d)[j]$, and frequency is changed to w_t.
- Update: In the case of $p = q$, $gM[gP(s,d)[j]][z].rank$ remains unchanged, and the

frequency value $gM[gP(s,d)[j]][z].c$ is added with w_t.Do not use the word "essentially" to mean "approximately" or "effectively".

- Do nothing: In the case of $p > q$, do nothing.

As shown in the part (D) of Figure 1, comparing $Rank(P(e_t))$ and $Rank(e_t)$, four operations on $P(e_t)$ are generated. '<' and '? 'means to evict and occupy; '=' means to update; '>' means to do nothing. In the case of eviction and update, the list attribute of the hash unit is updated while the rank and weight values are updated.

Algorithm 1: $\mathcal{PR}_B$-Sketch-Update $((s,d),w_t)$

```
for each row z = 1,2, ... , v do
  gPn←getgPattern(s, d);
  gRk←getgRank(s, d);
  for i = 0,1, ... , L − 1 do
    for j = 0,1, ... , L − 1 do
      (x, y) ← Pattern[i × L + j]; u = gM[x][y][z];
      if gRk[i × L + j] < u.rank then
        Update u with (w_t, gRk[i × L + j], [(s, d)]);
      else if gRk[i × L + j] = u.rank then
        Update u by adding w_t;
        Inserting (s, d) into u.list by with limit D;
```

C. Queries

Algorithm 2: $\mathcal{PR}_B$-Sketch-EdgeQuery(s,d)

```
1  r←a very large value;
2  for each row z = 1,2, ... , v do
3    gPn←getgPattern(s, d);
4    gRk←getgRank(s, d);
5    for i = 0,1, ... , L − 1 do
6      for j = 0,1, ... , L − 1 do
7        (x, y) ← gPn[i × L + j]; u = gM[x][y][z];
8        if gRk[i × L + j] = u.rank then
9          r_tmp ← u.c;
10   if u.list.length = 1 then
11     return (r_tmp,1);
12   else if u.list.length ≤ D then
13     len_tmp ← u.list.length;
14     for (s_l, d_l) in u.list & (s_l, d_l) ≠ (s, d) do
15       (r_l, b_l) ← sEdgeQuery( PR_B-Sketch, (s_l, d_l));
16       if b_l = 1 then
17         r_tmp−= r_l; len_tmp − −;
18         if len_tmp = 1 then
19           Return (r_tmp,1);
20   r = min (r, r_tmp);
21 Return (r,0);
```

1) Edge query

Edge query that returns the weight of the queried edge is the most basic query.$\mathcal{PR}_B$-Sketch first determines the pattern and rank of the queried edge (s,d). Then, for the $v \times L^2$ hash units in the v patterns, each hash unit with the same rank value as the corresponding value of the rank vector is selected and checked. Finally, the smallest weight value among these hash units is returned. The hash unit corresponding to the minimum value is the least conflicted, in other words it is the closest to the true weight of edge (s,d). At this time, if the length of the edge conflict list is 1, it means that the value is equal to the true value; otherwise, $\mathcal{PR}_B$-Sketch tries to query the true weight value by using the conflict list of each hash unit in v patterns. As shown in Algorithm **2**, when lines 12-19 are deleted from Algorithm **2**, it is a simple query without using the conflict list. Lines 12-19 optimize sEdgeQuery () by using the conflict list.

2) Reachability query.

The reachability query is a Boolean query. Given two nodes s and d, if there is a reachable path between s and d, return 1; otherwise, return 0. If there is a connected edge (s,d) between the two nodes, then there is a path reachable between nodes s and d; Otherwise, a depth-first traversal is performed to check whether there is a reachable path between s and d. We know that there are no false negatives for reachability queries on this structure, only false positives. If the result of the reachability query between s and d is 0, then there must be no reachable path between s and d; on the contrary, if the query result is 1, then there is still the possibility that the reachable path between *s* and *d* does not exist in real situations.

IV. RESULT AND DISCUSSIONS

This part introduces the experimental research results of $\mathcal{PR}_B$-Sketch. Based on two queries, edge query, and reachability query, $\mathcal{PR}_B$-Sketch, CMR, TCM and GSS are compared. In addition, the utilization of the hash space is estimated. All experiments are carried out on a server with four Intel(R) Xeon(R) E5-2640 2.40GHz CPUs of ten cores, 160GB memory, and 4 NVIDIA Tesla K40 cards. The operating system is Ubuntu 16.04. $\mathcal{PR}_B$-Sketch algorithms are implemented in eclipse and compiled using C and C++ code.

A. Data Set

We have selected three real-world datasets, all of which have been shown in Table 2. The first data set has the characteristics of unbalanced data stream frequency distributions. The data stream frequency distribution of the latter two data sets is uniform.

B. Metrics

Average Relative Error (ARE). Measure the accuracy of edge query. Given a query q, its relative error is:

$$RE(q) = \frac{f^{\wedge}(q)}{f(q)} - 1 \tag{4}$$

Among them, $f^{\wedge}(q)$ and $f(q)$ are the estimated result and the true result of query q respectively.

True Negative Recall (TNR). It measures the accuracy of reachability queries. The higher the TNR, the better.

Table 2. Dataset

DataSet	Edges	Nodes
web-NotreDame	1,497,134	325,729
Amazon	1,234,877	262,111
Wikitalk	5,021,410	2,394,385

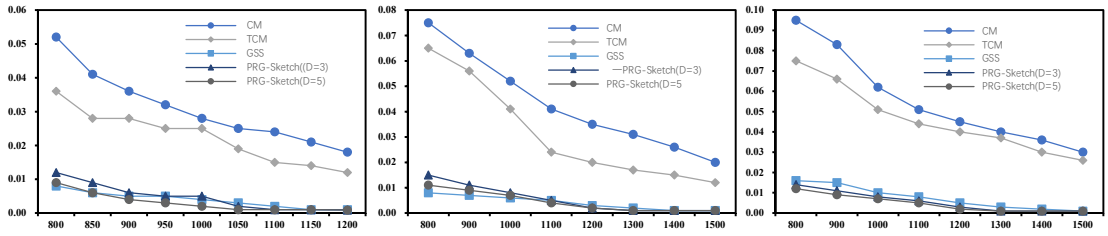
$$TNR(Q) = \frac{C^{\wedge}(Q)}{C(Q)} \tag{5}$$

Where $C^{\wedge}(Q)$ and $C(Q)$ are the estimated unreachable number and the true unreachable number in the combination of reachability query, respectively.

Hash Space Utilization. Measure the space utilization of the hash table. It is defined as the number of occupied units of the hash table divided by the total number of hash units.

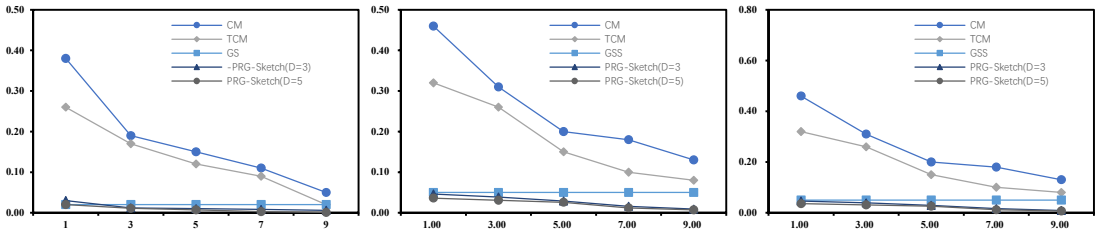
C. Edge Query

In this part, we evaluate the accuracy of CM, TCM, GSS and $\mathcal{PR}_B$-Sketch in edge query. Figure 2, Figure 3, Figure 6(a) show the average relative error of edge queries for the above three data sets when different parameters (hash table width, number of hash functions and pattern length) are changed. Results show that the accuracy of the edge query between the $\mathcal{PR}_B$-Sketch method and the GSS method is close, far better than TCM and CM.



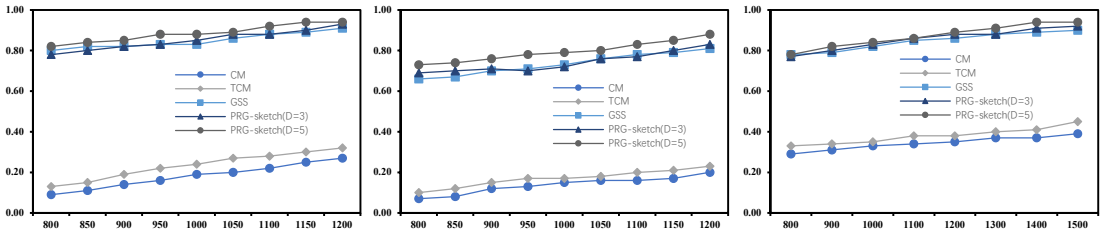
(a) Web-NotreName (ARE) (b) Amazon (ARE) (c) Wikitalk (ARE)

Figure 2. The average relative error of the edge query with different hash table widths.



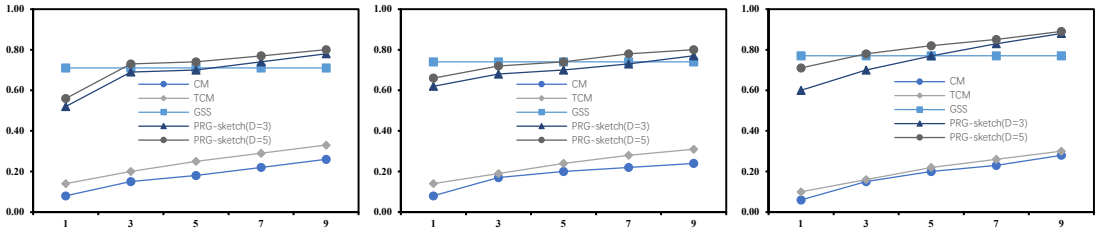
(a) Web-NotreName (ARE) (b) Amazon (ARE) (c) Wikitalk (ARE)

Figure 3. The average relative error of the edge query with different number of hash functions.



(a) Web-NotreName (TNR) (b) Amazon (TNR) (c) Wikitalk (TNR)

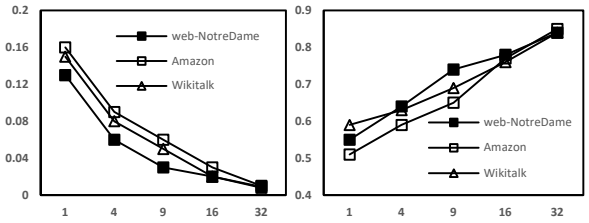
Figure 4. The true negative recall rate of the reachability query with different hash table widths.



(a) Web-NotreName (TNR) (b) Amazon (TNR) (c) Wikitalk (TNR)

Figure 5. The true negative recall rate of the reachability query with different number of hash functions.

In Figure 2, the number of hash functions of the three methods of CM, TCM and $\mathcal{PR}_B$-Sketch is set to 3, the pattern length in $\mathcal{PR}_B$-Sketch is 8, and the upper limit of the conflict list is 3. When the width of the hash table reaches a certain value, the edge query accuracy of $\mathcal{PR}_B$-Sketch is close to that of GSS.



(a)ARE (b)TNR

Figure 6. The ARE of the edge query and the TNR of the reachability query with different pattern lengths.

In Figure 3, the width of the hash table is set to 1000, the pattern length in $\mathcal{PR}_B$-Sketch is 9, and the upper limit of the conflict list is 3. With the increase of the number of hash functions, the relative error of the edge query of the other three methods, except GSS, is decreasing. The accuracy of GSS is not affected by the number of hash functions.

The basic parameter value setting in Figure 6(a) is the same as the above experiment; the pattern length is changed, and the accuracy of the edge query is estimated. As the pattern length increases, the hash units that can be occupied by the low-frequency edge increase, and the degree of overestimation decreases, which improves the query accuracy of the low-frequency edge, thus improving the overall edge query accuracy.

D. Reachability Query

In this experiment, we use web-NotreName, Amazon, and Wikitalk data sets to estimate the reachability based on TCM, GSS, and CM. The above graph summary methods can ultimately retain the reachability information in the original graph, but due to hash collisions, the unreachable edges in the original graph may be reachable by the above method. Therefore, we randomly generated 1000 unreachable node pairs in the original graph for each data set in this experiment to estimate the number of unreachable reachability queries through the above three methods (TCM, GSS, and CM). That is, the true negative recall rate (TNR) is used to measure the accuracy of the reachability query.

Figure 4, Figure 5, Figure 6(b) shows the reachability query true negative retrieval rate for three data sets with different parameters (hash table width, number of hash functions, and pattern length). It can be seen from the figures that as the width of the hash table, the number of hash functions and the length of the pattern increase, $\mathcal{PR}_B$ -Sketch can achieve better performance, and the TNR of $\mathcal{PR}_B$-Sketch and GSS is much higher than the TNR of TCM and CM.

E. Hash Space Utilization

This section evaluates the hash space utilization of the several methods mentioned above. As shown in Figure 7, the hash space utilization of the $\mathcal{PR}_B$-Sketch method is higher than that of the CM and TCM methods, close to GSS, and much higher than the TCM and CM methods. Because the pattern strategy of $\mathcal{PR}_B$-Sketch greatly improve the utilization of hash space.

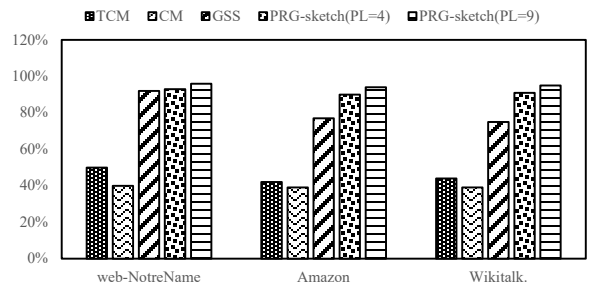


Figure 7. The Hash space utilization on different data sets.

V. CONCLUSIONS

In this article, we proposed $\mathcal{PR}_B$-Sketch, a graph data stream summary based on Pattern and Rank, which can support massive graph stream data. Our method is based on three real-world data sets and compared with existing graph data stream summarization methods. Experiments show that when the graph data stream is large, and the edge frequency distribution is unbalanced, $\mathcal{PR}_B$-Sketch shows superiority compared to other methods in query accuracy and hash space utilization. We will work hard to optimize this method in the future and find other methods that can replace storing the conflict list and improve query accuracy in the future.

REFERENCES

[1] G. Cormode, and S. Muthukrishnan, “An improved data stream summary: the count-min sketch and its applications,” vol. 55, no. 1 %J J. Algorithms, pp. 58–75, 2005.

[2] N. Tang, Q. Chen, and P. J. A. Mitra, “Graph Stream Summarization: From Big Bang to Big Crunch,” 2016.

[3] A. Khan, and C. Aggarwal, “Query-friendly compression of graph streams,” in Proceedings of the 2016 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, Davis, California, 2016, pp. 130–137.

[4] C. S. A. C, T. G. B, Y. G. A, H. Z. A, and X. Y. J. I. S. A, “Labeled graph sketches: Keeping up with real-time graph streams,” vol. 503, pp. 469-492, 2019.

[5] X. Gou, L. Zou, C. Zhao, and T. Yang, “Fast and Accurate Graph Stream Summarization,” 2018.

[6] F. Zhang, L. Zou, L. Zeng, and X. Gou, “Dolha - an efficient and exact data structure for streaming graphs,” *World Wide Web,* vol. 23, no. 2, pp. 873-903, 2020/03/01, 2020.

[7] N. Ashrafi-Payaman, M. R. Kangavari, S. Hosseini, and A. M. Fander, “GS4: Graph stream summarization based on both the structure and semantics,” *The Journal of Supercomputing,* vol. 77, no. 3, pp. 2713-2733, 2021/03/01, 2021.

[8] M. Chen, R. Zhou, H. Chen, and H. J. I. n. I. C. o. D. C. S. Jin, “Scube: Efficient Summarization for Skewed Graph Streams,” pp. 100-110, 2022.

[9] Z. Li, Z. Li, Z. Fan, J. Zhao, S.-L. Zeng, P. Luo, and K. J. E. Liu, “Cuckoo Matrix: A High Efficient and Accurate Graph Stream Summarization on Limited Memory,” 2023.

[10] Y. Jia, Z. Gu, Z. Jiang, C. Gao, and J. Yang, “Persistent graph stream summarization for real-time graph analytics,” *World Wide Web,* vol. 26, no. 5, pp. 2647-2667, 2023/09/01, 2023.

[11] Z. Li, S. Liu, J. Liu, Y. Zhang, T. Liang, and K. Liu, “SIM: A fast real-time graph stream summarization with improved memory efficiency and accuracy,” *Computer Networks*, pp. 110502, 2024/05/11/, 2024.

[12] R. Rubinfeld, and A. J. X. Guo, "6.842 Randomness and Computation."

[13] R. Motwani, and P. J. S. N. Raghavan, “Randomized algorithms,” vol. 26, pp. 48-50, 1995.