Computer Networks 198 (2021) 108403

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

DMatrix: Toward fast and accurate queries in graph stream

Changsheng Hou[1], Bingnan Hou[1], Tongqing Zhou *, Zhiping Cai *

College of Computer, National University of Defense Technology, Changsha, 410073, Hunan, China

ARTICLE INFO

Keywords:
Graph stream
Graph sketch
Approximate query

ABSTRACT

The graph stream has recently arisen in many interactive scenarios. Characterizing with large volume and high dynamic, graph streams are known to be difficult for high-speed summary and analysis, especially provided with limited resource availability. Existing solutions mainly use sketch-based methods to estimate the weight of items (e.g., Count-Min Sketch) and preserve the underlying graph structure information (e.g., TCM). Unfortunately, these solutions neither support complex graph-based queries nor achieve efficient real-time queries. In view of these limitations, we design DMatrix, a novel 3-dimensional graph sketch to facilitate fast and accurate queries in graph stream. Both structural query and weight-based estimation are supported with DMatrix. Through the integration of representative key reservation and majority voting, DMatrix can effectively narrow the error bounds of queries with real-time response efficiency. Both theoretical analysis and experimental results confirm that our solution is superior in accuracy and efficiency comparing with the state-of-the-art.

1. Introduction

The last decade has seen the massive adoption of graph streams for modeling complex structured data in interactive applications such as network traffic and social networks. As a time-evolving data sequence, a graph stream can continuously depict entities (e.g., social media users) and the connections among entities (e.g., interactions or friendships) and serves as the building block for a broad spectrum of services, including network anomaly detection [1], community discovery [2], service usage analysis [3] and etc.

Yet, analyzing large graph streams in real-time is in fact rather challenging due to its sheer volume and high changing velocity [4]. For example, it is reported that each link in large ISPs or data centers handles around millions of packets per second [5]. Such situations are worse when facing small and limited memory. As a result, traditional data structures (e.g., adjacency lists) are not appropriate for storing the graph streams.

Fortunately, fast and approximated answers, instead of the exact ones, are expected in the context of data streaming applications. To meet such requirements, sketches are widely used to generate synopses in a space-saving way for approximate weight estimation [6,7], finding heavy-hitters and mining top-k items [8–10]. Nevertheless, a direct adaptation of the aforementioned classical sketches over graph streams will lose the underlying structural information in the graph data.

We illustrate by an example that the limitations of classical sketches in various types of queries. Fig. 1 shows a sample graph stream and the sketches. The classical sketches shown in Fig. 1(b) and (c) can only be used as either a "node sketch" or an "edge sketch" which treats node labels (e.g., A and B) and edge labels (e.g., AB and CD) as hash keys, respectively. A node sketch maps each node to a bucket using the hash function h_{node}, which can summarize the weight of the outgoing traffic of each node as shown in Fig. 1(b). Similarly, an edge sketch maps each edge into a bucket that can estimate the approximate weight of each edge from the buckets as shown in Fig. 1(c). However, the classical sketches neither support node- and edge-weight queries simultaneously nor provide the structural information of the graph stream. In view of these limitations, graph sketch techniques (e.g., TCM [11] and gMatrix [12]) are proposed to summarizing graph streams in a generalized way as illustrate in Fig. 1(d). For example, the graph sketch maps the source node B and the destination node C to the first row (i.e., $h_{graph}(B) = 1$) and the first column (i.e., $h_{graph}(C) = 1$), respectively. The weight of the edge BC is estimated by the red bucket, i.e., 2. Meanwhile, the outgoing traffic weight of node B can be obtained by summing up the values in the first row i.e., 4. In addition, the graph sketch can also answer the subgraph weight query (e.g., the sum of the weight in subgraph that containing nodes A, B, and C.) and the path reachability query (e.g., whether node D reaches node E via

* Corresponding authors.
E-mail addresses: houchangsheng@nudt.edu.cn (C. Hou), houbingnan19@nudt.edu.cn (B. Hou), zhoutongqing@nudt.edu.cn (T. Zhou), zpcai@nudt.edu.cn (Z. Cai).

[1] Changsheng Hou and Bingnan Hou are the co-first authors.

https://doi.org/10.1016/j.comnet.2021.108403
Received 8 April 2021; Received in revised form 6 July 2021; Accepted 4 August 2021
Available online 14 August 2021

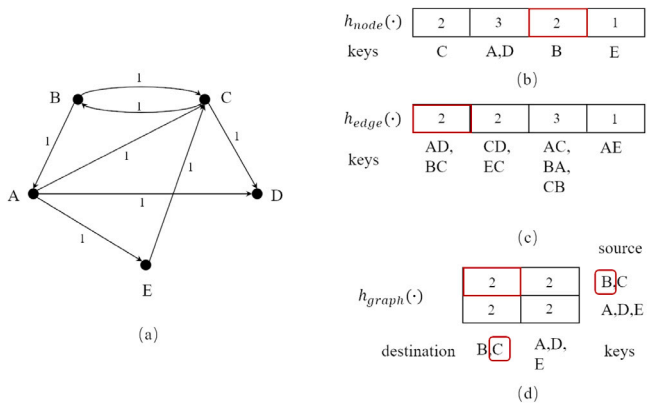

Fig. 1. A sample graph stream and the sketches. The red boxes highlight the nodes and edges we are concerned with and their mapping positions.

edges whose weight is larger than a threshold) that give the underlying structural information of the graph stream.

Although graph sketches can provide multi-dimensional queries, the estimation accuracy and query efficiency of these solutions are poor, which would be critical for timely and accurate data monitoring over graph stream. We are thus motivated to propose a modified graph sketch, called DMatrix, for real-time summary and analysis of graph stream by jointly considering estimation accuracy and query efficiency. Like TCM, DMatrix is designed to be general-purpose, namely, it can support common query operations, including weight-based queries (e.g., edge/node weight) and structure information queries (e.g., path reachability). In order to mitigate the accuracy limitations of existing graph sketches, we integrate a majority voting process in the 3-dimensional sketch structure for the recording of the most representative graph edge. By using a stream fashion in tracking the edge weight, such a modification is proved to be effective in narrow the error bounds of the weight-based queries. Since we reserve a field in each bucket for keys of interest, DMatrix can accomplish reversible query in real-time, which is required in timely streaming applications (e.g., anomalous traffic detection).

In a nutshell, DMatrix enables accurate and efficient queries in stream data with an acceptable consumption of storage resources. The main contributions are as follows:

1. We propose DMatrix, a novel 3-dimensional graph sketch for fast and accurate summary of graph streams. DMatrix retains both the structure- and weight-based information over the stream data.
2. We present theoretical analysis on DMatrix for its updating /querying time complexity and querying accuracy. Generally, DMatrix has smaller error bounds and lower time complexity compared with the state-of-the-art solutions.
3. We conduct experiments on three real-world stream datasets to evaluate the power of DMatrix in supporting different analyses. The results evaluate the superiority of DMatrix for graph stream in terms of both effectiveness and efficiency.

This paper is organized as follows: Section 2 discusses the background and related work. Section 3 introduces the preliminaries of the proposed sketch. Section 4 describes the design details of DMatrix. Section 5 presents the supported queries and gives a theoretical analysis of the performance. Section 6 presents the theoretical comparison with the state-of-the-art solution. Section 7 presents the experimental results on real-world datasets. Finally, Section 8 concludes this work.

2. Background and related work

We briefly review the relevant efforts devoted to data stream summarization from two aspects: classical sketches and graph sketches.

Sketches are stream data aggregation structures that track values in a fixed number of entries called buckets. Classical sketches (e.g., Count Sketch [13], K-ary Sketch [14], and Count-Min Sketch [15]) apply linear projections of the data with multiple hash functions into lower-dimensional spaces that preserve the aggregation features of the data. Let w denote the number of hash functions, a sketch is usually organized into a structure of $w \times h$ buckets. For each incoming key–value pair $(x, f; t)$ received in time-stamp t, the sketch hashes it w times using the pairwise independent hash functions, and each hash function maps the key to one of the h buckets. The bucket is regarded as a counter with an initial value of 0, which records the value corresponding to the key mapped to the bucket. Since hash collisions may cause multiple keys to be projected into the same bucket, the sketch can only estimate the value of x but not give an accurate one. For example, Count-Min Sketch takes the minimum value of w in all buckets as the estimated sum of key x. Note that classical sketches are designed in a two-dimensional bucket array, which prevents them from querying structure-based information (e.g., path- and subgraph-based queries).

Graph sketches [4,11,12,16] improve classical sketches for graph streams, aiming to summarize graph streams in a generalized way to support real-time updates and queries. gSketch [16] studies the summary construction and edge frequency of the graph stream. However, it is limited to queries based on edge frequency and cannot handle more complex queries based on graph structure. TCM [11] and GSS [4] aim at preserving the graph structure and support various types of queries. However, they cannot support real-time queries by reversing hashing technique, these solutions either need to traverse the entire key space to get the edge/node of interest or need to store an additional index table to record the keys and their hash values. gMatrix [12] recovers the key by using module hashing technology [17] to prune the key space. When recovering a key, gMatrix needs to enumerate each sub-key space and combine the recovered sub-keys to form the complete key. However, the time cost of traversing the sub-key space is expensive and it increases greatly with the key length. Another key recovery technique considers both Bloom Filter and invertibility [18], yet its sensitive to hash collisions which cannot recover the keys corresponding to a bucket if multiple keys are mapped to this bucket. In contrast, DMatrix aims at efficient recovery of the keys, which stores the key of interest in the bucket directly with little consumption of storage resources, so as to greatly reduce the computational cost of key recovery.

The types of query supported by different sketches are summarized in Table 1, which shows that DMatrix is much more general than other sketches.

3. Preliminaries

3.1. Graph stream summarization

A graph stream is a sequence of elements $e = (x, y, f; t)$ arrived in continuous time, where x, y are node identifiers and edge (x, y) with a weight/frequency of f is encountered at time-stamp t. The frequency of the edge can be regarded as an arriving edge with a weight of 1. Therefore, the term *weight* used in this paper covers the consideration of the weight and frequency of edges/nodes. Intuitively, the node label uniquely identifies a node, which is actually an identifier. And this identifier can be an IP address in network traffic data, or a user ID in social networks. Such a stream, $G = \langle e_1, e_2, \ldots, e_m \rangle$ naturally defines a graph $G = (V, E)$ where V and E are the sets of node and edge, respectively. In a directed graph, we define $\omega(x, y)$ the aggregated edge weight from node x to node y, and $\omega_{out}(x)$ ($\omega_{in}(x)$) the aggregated outgoing (incoming) weight of node x. The purpose of graph stream summarization is to design a suitable graph sketch $G_s = (V_s, E_s)$ to represent $G = (V, E)$, where the following condition hold [4,11]:

1. G_s is a graph;
2. $|G_s| \ll |G|$: the size of G_s is far less than G, preferably in sublinear space;

Table 1
Queries supported by different sketches.

Sketch	Edge weight	Node weight	Heavy-hitter edge	Heavy-hitter node	Heavy-changer edge	Heavy-changer node	Subgraph weight	Path reachability	Reversible query
DMatrix	✓	✓	✓	✓	✓	✓	✓	✓	✓
TCM [11]	✓	✓	✓	✓	✓	✓	✓	✓	✗
gMatrix [12]	✓	✓	✓	✓	✓	✓	✓	✓	✓
GSS [4]	✓	✗	✓	✗	✓	✗	✓	✓	✓
gSketch [16]	✓	✗	✓	✗	✓	✗	✓	✗	✗
Classical edge sketches	✓	✗	✓	✗	✓	✗	✓	✗	✗
Classical node sketches	✗	✓	✗	✓	✗	✓	✗	✗	✗

3. When a new edge in the graph stream arrives, G_s updates and the time complexity should be $O(1)$.
4. G_s supports various types of queries over the original streaming graph G with small margin of errors.

3.2. Categories of graph query

We classify the query operations over graph stream summarization into four categories as follows.

1. Edge-based query: *Edge weight query* which determines the weight of an edge (x, y); *Heavy-hitter edge query* which returns the edges whose weight is larger than a given threshold $\alpha(\alpha \in (0, 1))$ i.e., $\omega(x, y) > \alpha\mathcal{F}$ where $\mathcal{F}$ denotes the total sum of all weight. *Heavy-changer edge query* which returns the edges whose weight change is larger than a given threshold $\beta(\beta \in (0, 1))$ on the total absolute change across two adjacent epochs i.e., $|\omega(x, y; t) - \omega(x, y; t-1)| > \beta\mathcal{D}$ where $\mathcal{D}$ denotes the total absolute change weight between two adjacent epochs.
2. Node-based query: *Node weight query* which determines the outgoing (incoming) weight of a node x; *Heavy-hitter node query* which returns the nodes whose aggregate weight based on all outgoing/incoming edges is larger than a given threshold $\alpha(\alpha \in (0, 1))$; *Heavy-changer node query* which returns the nodes whose weight change is larger than a given threshold $\beta(\beta \in (0, 1))$ on the total absolute change across two adjacent epochs.
3. Subgraph-based query: *Subgraph weight query* which determines the aggregated weight of edges in a subgraph derived from a given subset of nodes V.
4. Path-based query: *Path reachability query* which finds whether the source node x can reach the destination node y via edges whose weight is larger than a given threshold $\gamma(\gamma \in (0, 1))$ on the total weight.

The graph stream summary enables these query tasks to be completed in sub-linear memory and time. The edge-based or node-based queries could be answered by the classical sketches, yet cannot answer both types of queries simultaneously. Graph sketches not only support various types of queries but also designed to be more appropriate to answer graph metrics (i.e., subgraph- and path-based query). One application of graph metrics, taking social networks as an example, the subgraph weight query which estimates the communication frequency within the community and the path reachability query can give the potential spreading path of a piece of news. Another use case in network monitoring tasks, for example, subgraph queries can locate certain topology structures in the dynamic networks and the reachability monitoring verifies the availability of network services. However, existing graph sketches cannot directly return the query for heavy-key (i.e., heavy-hitter and heavy-changer). Therefore, DMatrix is designed to be a general-purpose sketch, which is capable of answering all the above queries in graph streams.

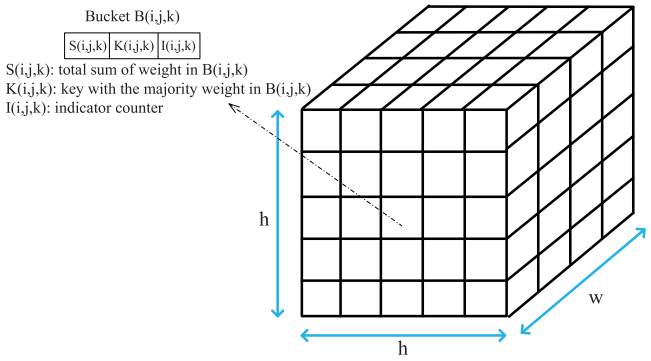

Fig. 2. Data structure of DMatrix.

4. The design of DMatrix

DMatrix is a novel graph sketch that supports various types of queries. The design goals are as following: (1) **Various queries**: DMatrix supports all the queries mentioned in Section 3.2. (2) **High accuracy**: DMatrix supports accurate estimation on weight-based queries with provable error bounds. (3) **High efficiency**: DMatrix is designed to be partly invertible, which can readily return all heavy-keys through the sketch structure itself instead of traversing the entire key space, thus improving the query efficiency. (4) **Limited memory**: DMatrix maintains compact data structures with small static memory allocation. For ease of reference, we list the notations in Table A.4.

4.1. Overview of DMatrix

DMatrix is designed as a three-dimensional sketch shown in Fig. 2. The first two dimensions represent the length and width of the bucket array and the third dimension corresponds to the depth of the bucket array. Each hash function defines a mapping from a node in set V to an integer in the range of $[1, h]$. DMatrix adopts the same value for length and width (i.e., h) and uses w pairwise-independent hash functions so that it is a 3-dimensional structure with $h \times h \times w$ buckets. Correspondingly, we define a bucket coordinate as $B(i, j, k)$, where i and j are indices of the hash-mapped nodes and k is the index of the hash functions being used for the mapping. Therefore, the kth hash function $h_k(\cdot)$ maps the edge (x, y) to the bucket index of $B(i, j, k)$, where $i = h_k(x)$, $j = h_k(y)$ and $k \in [1, w]$.

4.2. Structure of the bucket

To facilitate the reversible query, which refers to finding the key corresponding to a node or an edge that meets certain conditions, we reserve a space within the bucket for the storage of the key. Due to the hash collision, different edges may project into the same bucket. Thus there may be more than one key mapped to the same bucket. LD-Sketch [19] occasionally expands the associative key array in the bucket to hold more candidate keys, yet dynamic memory allocation is very costly. To save computing and storage overhead, DMatrix keeps

only one key in each bucket. For buckets with hash collisions, we will select one key to store in the bucket. In fact, each bucket contains an aggregation of edge weight with the same hash value, and the key we chose to store should best represent the aggregation result. To achieve this, we apply the majority vote algorithm (MJRTY) used in [20], which enables us to track the key with the majority weight in the bucket in an online streaming fashion. MJRTY [21] processes a stream of votes and aims to find the majority vote.

As shown in Fig. 2, $B(i,j,k)$ denotes the bucket at the ith row, the jth column and the kth depth, where $i,j \in [1,h], k \in [1,w]$. Each bucket $B(i,j,k)$ consists of three fields: (1) $S(i,j,k)$, which records the total value of all weight hashed to the bucket; (2) $K(i,j,k)$, which store the key with the majority weight in the bucket currently; and (3) $I(i,j,k)$, which is an indicator counter that decides whether to retain or replace the candidate key in $K(i,j,k)$. Thanks to MJRTY, which enables us to track the key with majority weight on evolving graphs such that DMatrix can hold the key with majority weight in the field K among the keys mapped to this bucket currently.

4.3. The updating process

The process of updating DMatrix is fairly straightforward as shown in Algorithm 1. We start off by initializing the three fields $S(i,j,k)$, $K(i,j,k)$ and $I(i,j,k)$ in the bucket to 0. For a incoming edge (x,y) whose weight is f, we first compute the index of the bucket $B(h_k(x), h_k(y), k)$. We then increment $S(h_k(x), h_k(y), k)$ by f (Line 2) and check whether (x,y) is stored in $K(h_k(x), h_k(y), k)$. If $K(h_k(x), h_k(y), k)$ equals (x,y), we increase $I(h_k(x), h_k(y), k)$ by f (Lines 3–4). Otherwise, we decrease $I(h_k(x), h_k(y), k)$ by f (Lines 5–6). If $I(h_k(x), h_k(y), k)$ drops below 0, we replace $K(h_k(x), h_k(y), k)$ by (x,y) and reset $I(h_k(x), h_k(y), k)$ with its absolute value (Lines 7–10).

Algorithm 1 Update DMatrix

Input: $(x, y, f; t)$
1: **for** $k = 1$ to w **do**
2: $S(h_k(x), h_k(y), k) \leftarrow S(h_k(x), h_k(y), k) + f$
3: **if** $K(h_k(x), h_k(y), k) == (x, y)$ **then**
4: $I(h_k(x), h_k(y), k) \leftarrow I(h_k(x), h_k(y), k) + f$
5: **else**
6: $I(h_k(x), h_k(y), k) \leftarrow I(h_k(x), h_k(y), k) - f$
7: **if** $I(h_k(x), h_k(y), k) < 0$ **then**
8: $K(h_k(x), h_k(y), k) \leftarrow (x, y)$
9: $I(h_k(x), h_k(y), k) \leftarrow -I(h_k(x), h_k(y), k)$
10: **end if**
11: **end if**
12: **end for**

5. Query operations and theoretical analysis

In the following, we introduce the query operations and provide their theoretical analysis.

5.1. Edge weight query

Algorithm 2 shows the query operation on edge weight. For a querying edge (x,y) hashed in w buckets, we calculate the estimate $\tilde{\omega}(x,y)_k$ of $\omega(x,y)_k$ in each depth of bucket array (Lines 1–7): if (x,y) and $K(h_k(x), h_k(y), k)$ are the same, we set $\tilde{\omega}(x,y)_k = (S(h_k(x), h_k(y), k) + I(h_k(x), h_k(y), k))/2$; otherwise, we set $\tilde{\omega}(x,y)_k = (S(h_k(x), h_k(y), k) - I(h_k(x), h_k(y), k))/2$ where $k \in [1,w]$. Finally, we return the minimum of all the $\tilde{\omega}(x,y)_k$ estimates as the final estimate $\tilde{\omega}(x,y)$ (Lines 8–9).

In this paper, we roughly estimate the time complexity by the number of bucket accesses brought by the query operation. Obviously, the time complexity of querying the edge weights is $\Theta(w)$.

Algorithm 2 Query for Edge Weight

Input: querying edge (x,y)
Output: estimate weight $\tilde{\omega}(x,y)$ of edge (x,y)
1: **for** $k = 1$ to w **do**
2: **if** $K(h_k(x), h_k(y), k) == (x, y)$ **then**
3: $\tilde{\omega}(x,y)_k \leftarrow (S(h_k(x), h_k(y), k) + I(h_k(x), h_k(y), k))/2$
4: **else**
5: $\tilde{\omega}(x,y)_k \leftarrow (S(h_k(x), h_k(y), k) - I(h_k(x), h_k(y), k))/2$
6: **end if**
7: **end for**
8: $\tilde{\omega}(x,y) \leftarrow min_{1 \le k \le w} \tilde{\omega}(x,y)_k$
9: **return** $\tilde{\omega}(x,y)$

Next, we show the probabilistic accuracy guarantee on edge weight query. The analysis assumes that DMatrix is configured with $w = log\frac{1}{\delta}$, $h = \sqrt{\frac{2}{\epsilon}}$, where $\epsilon \in (0,1)$ is the approximation parameter, $\delta \in (0,1)$ is the error probability, and the logarithm base is 2.

Theorem 1. *Given the current total edge weight $\mathcal{F}$, the edge weight estimation result satisfies $\omega(x,y) \le \tilde{\omega}(x,y) \le \omega(x,y) + \frac{\epsilon\mathcal{F}}{2}$ with a probability of at least $1-\delta$.*

Proof. From Algorithm 2, we can easily get that $\tilde{\omega}(x,y) \ge \omega(x,y)$. Next, we show the upper bound of $\tilde{\omega}(x,y)$. Consider the expectation of the total sum of all weight except (x,y) in its mapped bucket $B(i,j,k)$. It is given by $E[S(i,j,k) - \omega(x,y)] = E[\sum_{s \ne x, d \ne y, h_k(s)=h_k(x), h_k(d)=h_k(y)} \omega(s,d)] \le \frac{\mathcal{F}-\omega(x,y)}{h^2} \le \frac{\epsilon\mathcal{F}}{2}$. By Markov's inequality, we have

$$Pr[S(i,j,k) - \omega(x,y) \ge \epsilon\mathcal{F}] \le \frac{1}{2}. \tag{1}$$

We now consider the estimate $\tilde{\omega}(x,y)_k$ in different depth of bucket array according to Algorithm 2. If $K(i,j,k) == (x,y)$, then $\tilde{\omega}(x,y)_k - \omega(x,y) = \frac{S(i,j,k)+I(i,j,k)}{2} - \omega(x,y) \le \frac{S(i,j,k)-\omega(x,y)}{2}$. It is obvious that the bucket $B(i,j,k)$ has a hash collision when the inequality is formed. If $K(i,j,k) \ne (x,y)$, then $\tilde{\omega}(x,y)_k - \omega(x,y) = \frac{S(i,j,k)-I(i,j,k)}{2} - \omega(x,y) \le \frac{S(i,j,k)-\omega(x,y)}{2}$.

Combining both cases, we have $Pr[\tilde{\omega}(x,y)_k - \omega(x,y) \ge \frac{\epsilon\mathcal{F}}{2}] \le Pr[\frac{S(i,j,k)-\omega(x,y)}{2} \ge \frac{\epsilon\mathcal{F}}{2}] \le \frac{1}{2}$ due to (1).

Since $\tilde{\omega}(x,y)$ is the minimum of all $\tilde{\omega}(x,y)_k$ in w different buckets, we have $Pr[\tilde{\omega}(x,y) \le \omega(x,y) + \frac{\epsilon\mathcal{F}}{2}] = 1 - Pr[\tilde{\omega}(x,y) \ge \omega(x,y) + \frac{\epsilon\mathcal{F}}{2}] = 1 - Pr[\tilde{\omega}(x,y)_k \ge \omega(x,y) + \frac{\epsilon\mathcal{F}}{2}, \forall k] \ge 1 - (\frac{1}{2})^w = 1 - \delta$. □

5.2. Node weight query

In a directed graph, a node has an outgoing degree and an incoming degree. In this paper, we refer the sum of the weight in the outgoing (incoming) edge of the node as outgoing (incoming) weight denotes as $\omega_{out}(x)$ ($\omega_{in}(x)$).

Algorithm 3 shows the query operation on node weight. Note that we use outgoing weight to illustrate the query operation on node weight, and the incoming weight is consistent with it. For a querying node x hashed in ith row and kth depth of the bucket array, we first calculate the estimate $\tilde{\omega}_{out}(x)_k$ of $\omega_{out}(x)$ in each depth of bucket array (Lines 2–8): we compare $(x,\cdot)$ to the key field $(K(h_k(x), j, k))$ of each bucket in the $h_k(x)$th row on each depth of bucket array, where $\cdot$ denotes it can be any node identifier and $j \in [1,h]$. If $(x,\cdot) == K(h_k(x), j, k)$, we set $\tilde{\omega}_{out}(x)_{j,k} = (S(h_k(x), j, k) + I(h_k(x), j, k))/2$; otherwise, we set $\tilde{\omega}_{out}(x)_{j,k} = (S(h_k(x), j, k) - I(h_k(x), j, k))/2$ where $k \in [1,w]$ and $j \in [1,h]$. Next, we calculate the depth estimate $\tilde{\omega}_{out}(x)_k = \sum_{j=1}^{h} \tilde{\omega}_{out}(x)_{j,k}$ on each depth of the bucket array (Line 9). Finally, we return the minimum of all the $\tilde{\omega}_{out}(x)_k$ estimates as the final estimate $\tilde{\omega}_{out}(x)$ (Lines 11–12). Similarly, the incoming weight of a node can be calculated by the corresponding column of the bucket array.

Algorithm 3 Query for Node Weight (outgoing)

Input: querying node x
Output: estimate output weight $\tilde{\omega}_{out}(x)$ of node x
1: **for** $k = 1$ to w **do**
2: **for** $j = 1$ to h **do**
3: **if** $K(h_k(x), j, k) == (x, \cdot)$ **then**
4: $\tilde{\omega}_{out}(x)_{j,k} \leftarrow (S(h_k(x), j, k) + I(h_k(x), j, k))/2$
5: **else**
6: $\tilde{\omega}_{out}(x)_{j,k} \leftarrow (S(h_k(x), j, k) - I(h_k(x), j, k))/2$
7: **end if**
8: **end for**
9: $\tilde{\omega}_{out}(x)_k \leftarrow \sum_{j=1}^{h} \tilde{\omega}_{out}(x)_{j,k}$
10: **end for**
11: $\tilde{\omega}_{out}(x) \leftarrow min_{1 \le k \le w} \tilde{\omega}_{out}(x)_k$
12: **return** $\tilde{\omega}_{out}(x)$

The time complexity of querying the node's aggregated weights is $\Theta(hw)$.

Theorem 2 shows the probabilistic accuracy guarantee on node weight query. As the number of rows and columns are the same in DMatrix, the outgoing and incoming weight of the node have the same accuracy probabilistic guarantee. For convenience, we use $\tilde{\omega}(x)$ and $\omega(x)$ to denote the estimate and real outgoing/incoming weight of node x, respectively. The proof is given in Appendix B.1.

Theorem 2. *Given the current total edge weight $\mathcal{F}$, the node weight estimation result satisfies $\omega(x) \le \tilde{\omega}(x) \le \omega(x) + \frac{\sqrt{2\epsilon}\mathcal{F}}{2}$ with a probability of at least $1 - \delta$.*

5.3. Heavy-hitter edge query

Algorithm 4 shows query operation on heavy-hitter edge. To query the heavy-hitters in a given threshold α, we first calculate the total weight $\mathcal{F}$ at this epoch (Line 1). Then we check every bucket: if $S(i, j, k) \ge \alpha\mathcal{F}$, we let $(x, y) = K(i, j, k)$ and query $\tilde{\omega}(x, y)$ from Algorithm 2 (Lines 4–6). We return all detected heavy-hitter edges (x, y) which satisfy $\tilde{\omega}(x, y) \ge \alpha\mathcal{F}$.

Algorithm 4 Query for Heavy-hitter Edge

Input: heavy-hitter threshold α
Output: all detected heavy-hitter edges $\mathcal{E}_\alpha$
1: $\mathcal{F} = \sum_{i=1}^{h} \sum_{j=1}^{h} S(i, j, 1)$
2: $\mathcal{E}_\alpha \leftarrow \emptyset$
3: **for** $\forall i \in [1, h], \forall j \in [1, h], \forall k \in [1, w]$ **do**
4: **if** $S(i, j, k) \ge \alpha\mathcal{F}$ **then**
5: $(x, y) \leftarrow K(i, j, k)$
6: **if** $\tilde{\omega}(x, y) \ge \alpha\mathcal{F}$ **then**
7: $\mathcal{E}_\alpha \leftarrow \mathcal{E}_\alpha \cup (x, y)$
8: **end if**
9: **end if**
10: **end for**
11: **return** $\mathcal{E}_\alpha$

In the heavy-hitter edge query process, the time complexity of scanning DMatrix to calculate the total weight $\mathcal{F}$ is $\Theta(h^2w)$. Next, the process goes through each bucket to check if $S(i, j, k) \ge \alpha\mathcal{F}$ with a time complexity of $\Theta(h^2w)$. For each bucket meets the condition $S(i, j, k) \ge \alpha\mathcal{F}$, it does a edge query process according to Algorithm 2. Therefore, the total time complexity of querying the heavy-hitter edges is $\Theta(2h^2w) + \Omega(w|\mathcal{E}_\alpha|)$, where $|\mathcal{E}_\alpha|$ denotes the number of detected heavy-hitter edges.

Theorem 3 shows the error bounds of the heavy-hitter edge query in DMatrix. The proof is given in Appendix B.2.

Theorem 3. *Given that $\alpha \ge \epsilon$, the probability that DMatrix returns every heavy-hitter edge is at least $1 - \delta$. And falsely returns a non-heavy hitter with weight no more than $(\alpha - \frac{\epsilon}{2})\mathcal{F}$ with a probability at most δ.*

5.4. Heavy-hitter node query

Algorithm 5 shows query operation on heavy-hitter node, which we take heavy-hitter on outgoing weight of a node as the example. We first calculate the total weight $\mathcal{F}$ at this epoch in a given threshold α (Line 1). Then we check the sum value of each row in each depth at the bucket array: if $\sum_{j=1}^{h} S(i, j, k) \ge \alpha\mathcal{F}$, $\forall j \in [1, h]$, we let $(x, y) = K(i, j, k)$ and query $\tilde{\omega}_{out}(x)$ from Algorithm 3 (Lines 4–7). We return all detected heavy-hitter node x which satisfy $\tilde{\omega}(x) \ge \alpha\mathcal{F}$.

Algorithm 5 Query for Heavy-hitter Node (outgoing)

Input: heavy-hitter threshold α
Output: all detected heavy-hitter nodes $\mathcal{V}_\alpha$
1: $\mathcal{F} = \sum_{i=1}^{h} \sum_{j=1}^{h} S(i, j, 1)$
2: $\mathcal{V}_\alpha \leftarrow \emptyset$
3: **for** $\forall i \in [1, h], \forall k \in [1, w]$ **do**
4: **if** $\sum_{j=1}^{h} S(i, j, k) \ge \alpha\mathcal{F}$ **then**
5: **for** $j = 1$ to h **do**
6: $(x, \cdot) \leftarrow K(i, j, k)$
7: **if** $\tilde{\omega}_{out}(x) \ge \alpha\mathcal{F}$ **then**
8: $\mathcal{V}_\alpha \leftarrow \mathcal{V}_\alpha \cup x$
9: **end if**
10: **end for**
11: **end if**
12: **end for**
13: **return** $\mathcal{V}_\alpha$

The time complexity of calculating the total weight $\mathcal{F}$ is $\Theta(h^2w)$. Then, the process goes through each bucket to calculate the total row weight $\sum_{j=1}^{h} S(i, j, k)$ with a time complexity of $\Theta(h^2w)$. For each row meets the condition $\sum_{j=1}^{h} S(i, j, k) \ge \alpha\mathcal{F}$, it does node queries on each distinct node in this row according to Algorithm 3. Therefore, the total time complexity on heavy-hitter node query is $\Theta(2h^2w) + \Omega(hw|\mathcal{V}_\alpha|)$, where $|\mathcal{V}_\alpha|$ denotes the number of detected heavy-hitter nodes.

Theorem 4 shows the error bounds of the heavy-hitter node query in DMatrix. The proof is given in Appendix B.3.

Theorem 4. *Given that $\alpha \ge \sqrt{2\epsilon}$, the probability that DMatrix returns every heavy-hitter node is at least $1 - \delta$. For a non-heavy-hitter node whose weight is no more than $(\alpha - \frac{\sqrt{2\epsilon}}{2})\mathcal{F}$, the probability of falsely returning it is at most δ.*

5.5. Heavy-changer edge query

Algorithm 6 shows query operation on heavy-changer edge. To query the heavy-changers in a given threshold β, we first get all the distinct edges $\mathcal{K}_E$ from the bucket array at both the adjacent epochs and calculate the total change $\tilde{D}$ between the adjacent epochs (Lines 1–11). Note that $\tilde{D}$ is an estimate of the ground-truth weight change D. And $\tilde{D} \le D$, since the hash collisions will cancel out weight change in different directions.

Then we use the estimated maximum change of weight for our query to reduce the number of false negatives. Specifically, let $U(x, y)$ and $L(x, y)$ be the upper and lower bounds of $\omega(x, y)$, respectively. We set $U(x, y) = \tilde{\omega}(x, y)$ returned by Algorithm 2 (Lines 15–16). Also, we set $L(x, y) \leftarrow max_{1 \le k \le w}\{L(x, y)_k\}$, where $L(x, y)_k$ is set as follows:

Algorithm 6 Query for Heavy-changer Edge

Input: heavy-changer threshold β
Output: all detected heavy-changer edges $\mathcal{E}_\beta$

1: $\mathcal{K}_E \leftarrow \emptyset$, $\tilde{D} \leftarrow 0$
2: **for** $k = 1$ to w **do**
3: $\quad \tilde{D}_k \leftarrow 0$
4: $\quad$ **for** $i = 1$ to h **do**
5: $\quad\quad$ **for** $j = 1$ to h **do**
6: $\quad\quad\quad \mathcal{K}_E += K^{t-1}(i,j,k)$
7: $\quad\quad\quad \mathcal{K}_E += K^{t}(i,j,k)$
8: $\quad\quad\quad \tilde{D}_k += |S^{t-1}(i,j,k) - S^{t}(i,j,k)|$
9: $\quad\quad$ **end for**
10: $\quad$ **end for**
11: $\quad \tilde{D} \leftarrow max(\tilde{D}, \tilde{D}_k)$
12: **end for**
13: $\mathcal{E}_\beta \leftarrow \emptyset$
14: **for** (x,y) in $\mathcal{K}_E$ do **do**
15: $\quad U^t(x,y) \leftarrow \tilde{\omega}^t(x,y)$
16: $\quad U^{t-1}(x,y) \leftarrow \tilde{\omega}^{t-1}(x,y)$
17: $\quad$ **if** $U^t(x,y) \geq \beta\tilde{D}$ or $U^{t-1}(x,y) \geq \beta\tilde{D}$ **then**
18: $\quad\quad L^t(x,y) \leftarrow$ LowEstimate$((x,y))$
19: $\quad\quad L^{t-1}(x,y) \leftarrow$ LowEstimate$((x,y))$
20: $\quad\quad \tilde{D}(x,y) \leftarrow max\{|U^t(x,y) - L^{t-1}(x,y))|, |L^t(x,y) - U^{t-1}(x,y)|\}$
21: $\quad\quad$ **if** $\tilde{D}(x,y) \geq \beta\tilde{D}$ **then**
22: $\quad\quad\quad \mathcal{E}_\beta \leftarrow \mathcal{E}_\beta \cup (x,y)$
23: $\quad\quad$ **end if**
24: $\quad$ **end if**
25: **end for**
26: **return** $\mathcal{E}_\beta$
27: **function** LowEstimate$((x,y))$
28: $\quad$ **for** $k = 1$ to w **do**
29: $\quad\quad$ **if** $(x,y) == K(h_k(x), h_k(y), k)$ **then**
30: $\quad\quad\quad L(x,y)_k \leftarrow I(h_k(x), h_k(y), k)$
31: $\quad\quad$ **else**
32: $\quad\quad\quad L(x,y)_k \leftarrow 0$
33: $\quad\quad$ **end if**
34: $\quad$ **end for**
35: $\quad L(x,y) \leftarrow max_{1\leq k\leq w}\{L(x,y)_k\}$
36: $\quad$ **return** $L(x,y)$
37: **end function**

for each hashed bucket $B(i,j,k)$ of (x,y), if $K(i,j,k) == (x,y)$ we set $L(x,y)_k = I(i,j,k)$, otherwise, we set $L(x,y)_k = 0$, where $h_k(x) = i, h_k(y) = j$ and $k \in [1,w]$ (Lines 27–37). Now, let $U^{t-1}(x,y)$ and $L^{t-1}(x,y)$ (resp. $U^t(x,y)$ and $L^t(x,y)$) be the upper and lower bounds of $\omega(x,y)$ in the previous (resp. current) epoch, respectively. The estimated maximum change of (x,y) on edge weight is given by $\tilde{D}(x,y) \leftarrow max\{|U^t(x,y) - L^{t-1}(x,y)|, |L^t(x,y) - U^{t-1}(x,y)|\}$ (Line 20).

Finally, we check every edge appears in the two adjacent epochs. For each edge, if $U^t(x,y) \geq \beta\tilde{D}$ or $U^{t-1}(x,y) \geq \beta\tilde{D}$, we estimate its weight change and return (x,y) as a heavy-changer edge if $\tilde{D}(x,y) \geq \beta\tilde{D}$.

The time complexity of calculating total weight change between two adjacent epochs is $\Theta(2h^2w)$. Then, the process estimates the upper bound weight (i.e., $U(x,y)$) of every distinct edge which appears in the two adjacent epochs with a total time complexity of $\Theta(2w|\mathcal{K}|)$, where $|\mathcal{K}_E|$ denotes the number of distinct edges appear in the two adjacent epochs. For each edge meets the condition $U^t(x,y) \geq \beta\tilde{D}$ or $U^{t-1}(x,y) \geq \beta\tilde{D}$, we will calculate the lower bound weight (i.e., $L(x,y)$) with a time complexity of $2w$ for each. Therefore, the total time complexity on heavy-changer edge query is $\Theta(2h^2w) + \Omega(4w|\mathcal{E}_\beta|)$, where $|\mathcal{E}_\beta|$ denotes the number of detected heavy-changer edges.

Theorem 5 shows the accurate guarantee of heavy-changer edge query. The proof is given in Appendix B.4.

Theorem 5. *Let $\mathcal{F}^t$ and $\mathcal{F}^{t-1}$ denote the total sum of all weight in the current and previous epochs. Given that $\frac{\beta\tilde{D}}{\epsilon} \geq max\{\mathcal{F}^t, \mathcal{F}^{t-1}\}$ and $\beta \geq \epsilon$, the probability that DMatrix returns every heavy-changer edge is at least $1-\delta$. For a non-heavy-changer edge whose weight is no more than $\beta\tilde{D} - \epsilon(\mathcal{F}^t + \mathcal{F}^{t-1})$, the probability of falsely returning it is at most $1-(1-\delta)^2$.*

5.6. Heavy-changer node query

Algorithm 7 shows query operation on heavy-changer node. To query the heavy-changers in a given threshold β, we first get all the distinct nodes $\mathcal{K}_V$ stored in the key field of the bucket array at both the adjacent epochs and calculate the total change $\tilde{D}$ between the adjacent epochs (Lines 1–11).

Then, we use the query results to reduce false negatives. The method of reducing false negatives in heavy node query is similar to heavy edge. Similarly, $\tilde{D}(x) \leftarrow max\{|U^t(x) - L^{t-1}(x)|, |L^t(x) - U^{t-1}(x)|\}$ gives the estimated maximum change of x on outgoing weight (Line 20).

Finally, we check every node appears in the two adjacent epochs. For each node, if $U^t(x) \geq \beta\tilde{D}$ or $U^{t-1}(x) \geq \beta\tilde{D}$, we estimate its weight change and return x as a heavy-changer node if $\tilde{D}(x) \geq \beta\tilde{D}$.

The time complexity of calculating total weight change between two adjacent epochs is $\Theta(2h^2w)$. Then, the process estimates the upper bound weight (i.e., $U(x)$) of every distinct node which appears in the two adjacent epochs with a total time complexity of $\Theta(2hw|\mathcal{K}_V|)$, where $|\mathcal{K}_V|$ denotes the number of distinct nodes appear in the two adjacent epochs. For each node meets the condition $U^t(x) \geq \beta\tilde{D}$ or $U^{t-1}(x) \geq \beta\tilde{D}$, we will calculate the lower bound weight (i.e., $L(x)$) with a time complexity of $2hw$ for each. Therefore, the total time complexity on heavy-changer node query is $\Theta(2h^2w) + \Omega(4hw|\mathcal{V}_\beta|)$, where $|\mathcal{V}_\beta|$ denotes the number of detected heavy-changer nodes.

Theorem 6 shows the accurate guarantee of heavy-changer node query. Please refer to Appendix B.5 for the proof.

Theorem 6. *Given that $\frac{\beta\tilde{D}}{\sqrt{2\epsilon}} \geq max\{\mathcal{F}^t, \mathcal{F}^{t-1}\}$ and $\beta \geq \sqrt{2\epsilon}$, the probability that DMatrix returns every heavy-changer node is at least $1-\delta$. For a non-heavy-changer node whose weight is no more than $\beta\tilde{D} - \sqrt{2\epsilon}(\mathcal{F}^t + \mathcal{F}^{t-1})$, the probability of falsely returning it is at most $1-(1-\delta)^2$.*

5.7. Subgraph weight query

The subgraph weight query is to compute the aggregated weight in a subgraph induced by a given subset $\mathcal{V}_s = \{x_1, \ldots, x_n\}$ of all nodes V in graph streams, where $\mathcal{V}_s \subseteq V$. Since we cannot grasp the true connectivity of the subgraph, we assume that there is a connection between each node in the subgraph. And all the edges in the subgraph is denoted by $\mathcal{E}_s = \{e_1, \ldots, e_m\}$ where $|\mathcal{E}_s| = \frac{|\mathcal{V}_s|(|\mathcal{V}_s|-1)}{2}$, $|\mathcal{E}_s|$ and $|\mathcal{V}_s|$ denote the number of edges and nodes in the subgraph, respectively.

Algorithm 8 shows the query operation on subgraph weight. We first get all the possible edges $\mathcal{E}_s$ in the subgraph (Lines 2–4). Then we estimate the weight of each edge in $\mathcal{E}_s$ according to Algorithm 1 and sum its estimated weight together to get the aggregated weight estimation of subgraph $\omega(\mathcal{V}_s)$ (Lines 5–8).

The time complexity of estimating the aggregated weight in a subgraph is $\Theta(w|\mathcal{E}_s|) = \Theta(w\frac{|\mathcal{V}_s|(|\mathcal{V}_s|-1)}{2})$, where $|\mathcal{V}_s|$ is the number of nodes in the subgraph.

Theorem 7. *$\omega(\mathcal{V}_s) \leq \tilde{\omega}(\mathcal{V}_s) \leq \omega(\mathcal{V}_s) + \frac{\epsilon\mathcal{F}|\mathcal{V}_s|(|\mathcal{V}_s|-1)}{4}$ with a probability at least $(1-\delta)^{\frac{|\mathcal{V}_s|(|\mathcal{V}_s|-1)}{2}}$, where $\mathcal{F}$ is the total edge weight in current and $|\mathcal{V}_s|$ denotes the number of nodes in the given subgraph.*

Theorem 7 shows the probabilistic accuracy guarantee on subgraph weight query. The proof of it is similar to Theorem 1, while it aggregates each edge weight estimation and treats them independently.

Algorithm 7 Query for Heavy-changer Node (outgoing)

Input: heavy-changer threshold β
Output: all detected heavy-changer nodes $\mathcal{V}_\beta$

$\mathcal{K}_V \leftarrow \emptyset$, $\tilde{D} \leftarrow 0$
for $k = 1$ to w **do**
  $\tilde{D}_k \leftarrow 0$
  for $i = 1$ to h **do**
    for $j = 1$ to h **do**
      $\mathcal{K}_V += K^{t-1}(i,j,k)$
      $\mathcal{K}_V += K^{t}(i,j,k)$
      $\tilde{D}_k += |S^{t-1}(i,j,k) - S^{t}(i,j,k)|$
    end for
  end for
  $\tilde{D} \leftarrow max(\tilde{D}, \tilde{D}_k)$
end for
$\mathcal{V}_\beta \leftarrow \emptyset$
for x in $\mathcal{K}_V$ do **do**
  $U^t(x) \leftarrow \tilde{\omega}^t_{out}(x)$
  $U^{t-1}(x) \leftarrow \tilde{\omega}^{t-1}_{out}(x)$
  if $U^t(x) \geq \beta\tilde{D}$ or $U^{t-1}(x) \geq \beta\tilde{D}$ **then**
    $L^t(x,y) \leftarrow$ LowEstimate(x)
    $L^{t-1}(x) \leftarrow$ LowEstimate(x)
    $\tilde{D}(x) \leftarrow max\{|U^t(x) - L^{t-1}(x))|, |L^t(x) - U^{t-1}(x)|\}$
    if $\tilde{D}(x) \geq \beta\tilde{D}$ **then**
      $\mathcal{V}_\beta \leftarrow \mathcal{V}_\beta \cup x$
    end if
  end if
end for
return $\mathcal{V}_\beta$
function LowEstimate(x)
  for $k = 1$ to w **do**
    for $j = 1$ to h **do**
      if $(x,\cdot) == K(h_k(x),j,k)$ **then**
        $L(x)_{j,k} \leftarrow I(h_k(x),j,k)$
      else
        $L(x)_{j,k} \leftarrow 0$
      end if
    end for
    $L(x)_k = \sum_{j=1}^{j=h} L(x)_{j,k}$
  end for
  $L(x) \leftarrow max_{1\leq k\leq w}\{L(x)_k\}$
  return $L(x)$
end function

Algorithm 8 Query for Subgraph Weight

Input: node set $\mathcal{V}_s = x_1, \ldots, x_n$
Output: estimate aggregated weight $\tilde{\omega}(\mathcal{V}_s)$ of node set $\mathcal{V}_s$

$\tilde{\omega}(\mathcal{V}_s) \leftarrow 0$, $\mathcal{E}_s \leftarrow \emptyset$
for $x_i, x_j \in \mathcal{V}_s, x_i \neq x_j$ **do**
  $\mathcal{E}_s += (x_i, x_j)$
end for
for (s,t) in $\mathcal{E}_s$ **do**
  $\tilde{\omega}(\mathcal{V}_s) += \tilde{\omega}(s,t)$
end for
return $\tilde{\omega}(\mathcal{V}_s)$

5.8. Path reachability query

We define the path reachability query to determine if a source node x is reachable to a destination node y via edges that have weight at least $\gamma\mathcal{F}$, where $\mathcal{F}$ is the total weight. This query returns a boolean result to tell whether there exists a path from x to y, s.t. the edge(s) on the path has/have a weight at least $\gamma\mathcal{F}$ for each. Like the heavy-key query, this query is also based on the partly reversible property of DMatrix, since we only want to get the connected edge information via the data structure itself, without adding additional storage space for reachability information. We note that this query is fairly straightforward to resolve with the use of heavy-hitter edge query. Algorithm 9 shows the query operation on path reachability. In the first step, we get all edges that have the weight of at least $\gamma\mathcal{F}$ according to Algorithm 4 (Lines 1–2). Then, we utilize the breadth first search algorithm (BFS) to answer the reachability question using these edges (Line 3).

Algorithm 9 Query for Path Reachability

Input: source node x, destination node y, weight threshold γ
Output: one path from x to y if exists, otherwise none

1: $\alpha \leftarrow \gamma$
2: $\mathcal{E}_\alpha \leftarrow$ Heavy-hitter Edge Query(α)
3: $p \leftarrow$ BFS$(x, y, \mathcal{E}_\alpha)$
4: **return** p

The time complexity of path reachability query contains two parts, i.e., the heavy-hitter edge query with time complexity of $\Theta(2h^2w) + \Omega(w|\mathcal{E}_\alpha|)$ and the BFS process. The roughly total time complexity of path query is $\Theta(2h^2w) + \Omega(w|\mathcal{E}_\alpha|)$.

Theorem 8. *DMatrix determines that there is a path from x to y, if x is reachable to y, with a probability at least $(1-\delta)^n$, provided that $\gamma \geq \epsilon$, where n is the number of edges the path contains. And falsely returns a nonexistent path with a probability at most $1-(1-\delta)^n$, s.t. each edge weight on the nonexistent path is no more than $(\gamma - \frac{\epsilon}{2})\mathcal{F}$.*

Theorem 8 shows the probabilistic accuracy guarantee on path query. The proof of it is similar to Theorem 3, except that it extends the proof from 1 to n heavy-hitter edges.

6. Theoretical comparison with TCM

We compare the probabilistic accuracy guarantee, the time complexity and the memory overhead of DMatrix with TCM [11], the state-of-the-art in graph sketch. Table 2 provides a summary of comparison with TCM on each query process, in terms of ϵ, δ, h, w. The proofs of probabilistic guarantee on TCM are given in Appendix C.

The weight-based querying operations (i.e., queries on edge weight, node weight and subgraph weight) show that the upper error bound of DMatrix is only half as TCM's with the same probabilistic guarantee and time complexity.

In the comparison of heavy-key and reachability queries, we study the metrics of true positive probability ($Pr[TP]$) and false positive probability ($Pr[FP]$) under certain constraints. $Pr[TP]$ and $Pr[FP]$ denote the probability of reporting a heavy-key/reachable path and the probability of falsely returning a non-heavy-key/infeasible path as a heavy-key/reachable path, respectively. Note that the true positive probability of heavy-hitter query in TCM is 1, which is higher than that of it in DMatrix, yet with orders of magnitude higher time complexity. As it has to query the whole key space to get all possible heavy-keys.

In short, DMatrix narrows the error bounds of weight-based queries by MJRTY and its reversibility greatly reduces the time complexity of heavy-key and reachability queries. Besides, its additional storage consumption (i.e., the key and indicator fields for each bucket) is acceptable when compared to TCM.

7. Experimental evaluation

We perform a number of experiments to compare the performance between DMatrix and TCM [11] in terms of accuracy and efficiency. We first give a brief description of our real-world datasets. Then we

Table 2
The probabilistic guarantee and time complexity of DMatrix and TCM.

Query	Sketch	Boundary	Probabilistic guarantee	Time complexity
Edge weight	DMatrix	$\omega(x,y) \le \tilde{\omega}(x,y) \le \omega(x,y) + \frac{\epsilon \mathcal{F}}{2}$	$\ge 1-\delta$	$\Theta(w)$
	TCM	$\omega(x,y) \le \tilde{\omega}(x,y) \le \omega(x,y) + \epsilon \mathcal{F}$	$\ge 1-\delta$	$\Theta(w)$
Node weight	DMatrix	$\omega(x) \le \tilde{\omega}(x) \le \omega(x) + \frac{\sqrt{2\epsilon}\mathcal{F}}{2}$	$\ge 1-\delta$	$\Theta(hw)$
	TCM	$\omega(x) \le \tilde{\omega}(x) \le \omega(x) + \sqrt{2\epsilon}\mathcal{F}$	$\ge 1-\delta$	$\Theta(hw)$
Heavy-hitter edge	DMatrix	$Pr[TP]$ s.t. $\alpha \ge \epsilon$ $Pr[FP]$ s.t. $\omega(s,t) \le (\alpha - \frac{\epsilon}{2})\mathcal{F}$	$\ge 1-\delta$ $\le \delta$	$\Theta(2h^2w) + \Omega(w\lvert\mathcal{E}_\alpha\rvert)$
	TCM	$Pr[TP]$ $Pr[FP]$ s.t. $\omega(s,t) \le (\alpha - \epsilon)\mathcal{F}$	1 $\le \delta$	$\Theta(w(2h^2 + \lvert E\rvert))$
Heavy-hitter node	DMatrix	$Pr[TP]$ s.t. $\alpha \ge \sqrt{2\epsilon}$ $Pr[FP]$ s.t. $\omega(s) \le (\alpha - \frac{\sqrt{2\epsilon}}{2})\mathcal{F}$	$\ge 1-\delta$ $\le \delta$	$\Theta(2h^2w) + \Omega(hw\lvert\mathcal{V}_\alpha\rvert)$
	TCM	$Pr[TP]$ $Pr[FP]$ s.t. $\omega(s) \le (\alpha - \sqrt{2\epsilon})\mathcal{F}$	1 $\le \delta$	$\Theta(hw(2h + \lvert V\rvert))$
Heavy-changer edge	DMatrix	$Pr[TP]$ s.t. $\frac{\beta\tilde{D}}{\epsilon} \ge max\{\mathcal{F}^t, \mathcal{F}^{t-1}\}$ & $\beta \ge \epsilon$ $Pr[FP]$ s.t. $\tilde{D}(s,t) \le \beta\tilde{D} - \epsilon(\mathcal{F}^t + \mathcal{F}^{t-1})$	$\ge 1-\delta$ $\le 1-(1-\delta)^2$	$\Theta(2h^2w) + \Omega(4w\lvert\mathcal{E}_\beta\rvert)$
	TCM	$Pr[TP]$ s.t. $D(x,y) \ge \beta\tilde{D} + \epsilon \cdot max\{\mathcal{F}^t, \mathcal{F}^{t-1}\}$ $Pr[FP]$ s.t. $D(s,t) \le \beta\tilde{D} - \epsilon \cdot max\{\mathcal{F}^t, \mathcal{F}^{t-1}\}$	$\ge (1-\delta)^2$ $\le 1-(1-\delta)^2$	$\Theta(2w(h^2 + \lvert E\rvert))$
Heavy-changer node	DMatrix	$Pr[TP]$ s.t. $\frac{\beta\tilde{D}}{\sqrt{2\epsilon}} \ge max\{\mathcal{F}^t, \mathcal{F}^{t-1}\}$ & $\beta \ge \sqrt{2\epsilon}$ $Pr[FP]$ s.t. $D(s) \le \beta\tilde{D} - \sqrt{2\epsilon}(\mathcal{F}^t + \mathcal{F}^{t-1})$	$\ge 1-\delta$ $\le 1-(1-\delta)^2$	$\Theta(2h^2w) + \Omega(4hw\lvert\mathcal{V}_\beta\rvert)$
	TCM	$Pr[TP]$ s.t. $D(x) \ge \beta\tilde{D} + \sqrt{2\epsilon} \cdot max\{\mathcal{F}^t, \mathcal{F}^{t-1}\}$ $Pr[FP]$ s.t. $D(s) \le \beta\tilde{D} - \sqrt{2\epsilon} \cdot max\{\mathcal{F}^t, \mathcal{F}^{t-1}\}$	$\ge (1-\delta)^2$ $\le 1-(1-\delta)^2$	$\Theta(2hw(h + \lvert V\rvert))$
Subgraph weight	DMatrix	$\omega(\mathcal{V}_s) \le \tilde{\omega}(\mathcal{V}_s) \le \omega(\mathcal{V}_s) + \frac{\epsilon\mathcal{F}\lvert\mathcal{V}_s\rvert(\lvert\mathcal{V}_s\rvert-1)}{4}$	$\ge (1-\delta)^{\frac{\lvert\mathcal{V}_s\rvert(\lvert\mathcal{V}_s\rvert-1)}{2}}$	$\Theta(w^{\frac{\lvert\mathcal{V}_s\rvert(\lvert\mathcal{V}_s\rvert-1)}{2}})$
	TCM	$\omega(\mathcal{V}_s) \le \tilde{\omega}(\mathcal{V}_s) \le \omega(\mathcal{V}_s) + \frac{\epsilon\mathcal{F}\lvert\mathcal{V}_s\rvert(\lvert\mathcal{V}_s\rvert-1)}{2}$	$\ge (1-\delta)^{\frac{\lvert\mathcal{V}_s\rvert(\lvert\mathcal{V}_s\rvert-1)}{2}}$	$\Theta(w^{\frac{\lvert\mathcal{V}_s\rvert(\lvert\mathcal{V}_s\rvert-1)}{2}})$
Path reachability	DMatrix	$Pr[TP]$ s.t. $\gamma \ge \epsilon$ $Pr[FP]$ s.t. $\exists(s,t) \in p, (s,t) \le (\gamma - \frac{\epsilon}{2})\mathcal{F}$	$\ge (1-\delta)^n$ $\le 1-(1-\delta)^n$	$\Theta(2h^2w) + \Omega(w\lvert\mathcal{E}_\alpha\rvert)$
	TCM	$Pr[TP]$ $Pr[FP]$ s.t. $\exists(s,t) \in p, (s,t) \le (\gamma - \epsilon)\mathcal{F}$	1 $\le 1-(1-\delta)^n$	$\Theta(\lvert E\rvert)$

$\epsilon, \delta \in (0,1)$ are the approximation parameter and error probability, respectively; $h(= \sqrt{\frac{2}{\epsilon}})$ and $w(= log\frac{1}{\delta})$ are the width/length and depth of the bucket array in DMatrix and TCM, respectively; $|\mathcal{E}_\alpha|$, $|\mathcal{V}_\beta|$, $|E|$ and $|V|$ are the number of heavy-hitter edges, heavy-hitter nodes, total edges and total nodes, respectively, where $|E| \gg |\mathcal{E}_\alpha|$ and $|V| \gg |\mathcal{V}_\beta|$.

introduce our evaluation criteria and parameter configurations used in the experiments.

All experiments are performed on a Linux platform with an Intel Core i9-9900KF CPU (3.60 GHz) and 32 GB DRAM memory, running Ubuntu. We provide the source code of DMatrix and publish the generated Twitter-communication stream dataset (at https://github.com/houchangsheng/DMatrix) so that others can build upon our work.

7.1. Datasets

We choose three real-world datasets. Details of the datasets are described as follows:

IP-trace stream. We use MAWI IP-trace dataset [22] which are all collected from the real-world network. Each trace is a pcap file providing raw packets that were captured for 15 min every day, since 2001 until now, on a trans-Pacific link between Japan and the US. The dataset is daily updated to include new traffic traces. In the experiments, we obtain the IP-traces captured on June 11, 2020. The traces are divided into five epochs in units of three minutes, which contains 22.7 million packets on average.

Twitter-communication stream. The Twitter graph dataset [23] consists of all public tweets during a seven-month period from June 1, 2009, to December 31, 2009. Each edge in the graph dataset represents a communication between two users in the form of a re-tweet. The edge is weighted by the number of communications between the corresponding source and destination users. For user privacy consideration, Twitter anonymizes this dataset and the re-tweets between users are only recorded once. To make the dataset has time-evolving characteristics, we simulate the process of tweeting between users in the real world. We first extracted the beginning 24 million user pairs in the dataset. Then we copied each user pair 1 to 9 times and assigned values to these records in a Pareto distribution. Finally, we arranged them out of order to form the dataset with a total of nearly 120M records. The reason why we utilized Pareto distribution to assign values to records is that Twitter and other web application data are always well-modeled by Pareto or its variant distribution [24].

DBLP co-author stream. We derived the author-pairs data from the latest DBLP archive [25]. Regarding 2,630,745 authors as nodes and 29,863,639 author-pairs as edges with the weight of 1 for each edge, indicating a co-authorship. With the gradual increase of the cooperation between authors, the co-authorship between them shows the time-evolving characteristics. Similar to IP-trace and Twitter-communication datasets, we divided the entire co-author stream data into five pieces evenly to facilitate the estimation of changes (i.e., heavy-changer) in authors' cooperation.

7.2. Metrics

Average relative error. This measure is used for the set of weight-based queries that return estimated weight, i.e., node, edge and subgraph weight queries. Given a graph stream $G = (V, E)$ received by now, the relative error of edge weight query is formalized as:

$$ARE(E) = \frac{\sum_{\forall(x,y)\in E} \frac{|\tilde{\omega}(x,y) - \omega(x,y)|}{\omega(x,y)}}{|E|},$$

where E denotes the edge set received by now.

Precision, recall and F1-score. The three metrics are used for heavy-key and reachability queries. Precision gives the fraction of true heavy-keys/paths reported over all reported heavy-keys/paths. Recall gives the fraction of true heavy-keys/paths reported over all true heavy-keys/paths. F1-score considers both precision and recall, which can be regarded as the harmonic average of precision and recall.

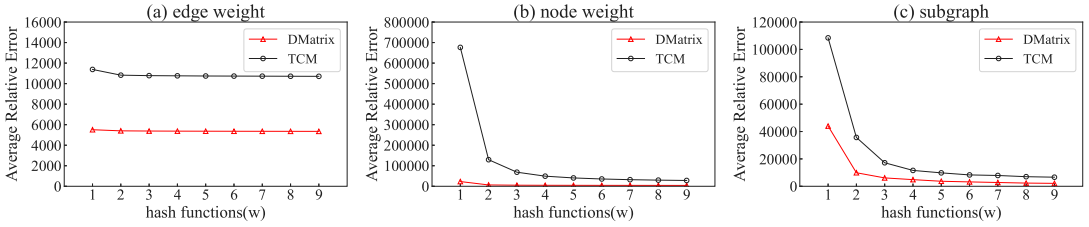

Fig. 3. Average relative error of weight-based queries with fixed compression rate (1/700) and varying w on IP-trace dataset.

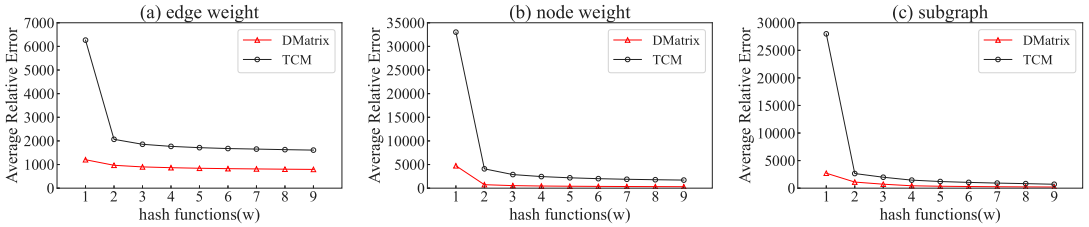

Fig. 4. Average relative error of weight-based queries with fixed compression rate (1/160) and varying w on Twitter-communication dataset.

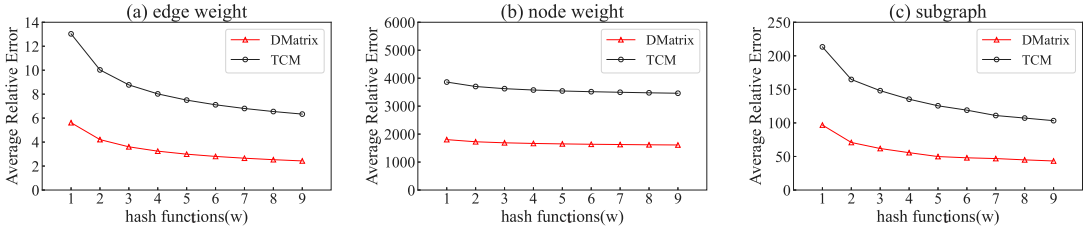

Fig. 5. Average relative error of weight-based queries with fixed compression rate (1/40) and varying w on DBLP dataset.

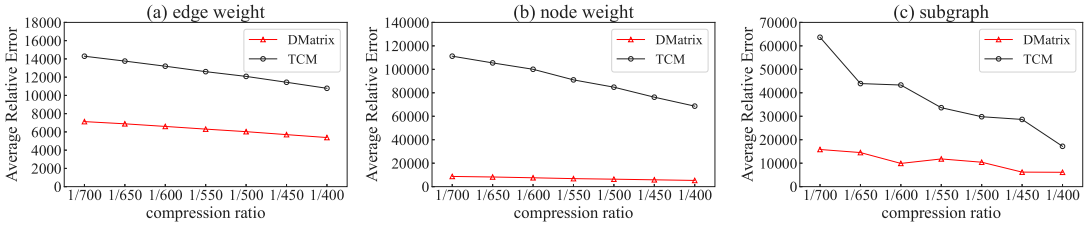

Fig. 6. Average relative error of weight-based queries with fixed $w = 3$ and varying compression rates on IP-trace dataset.

Average query time. The average query time compares the average time cost of a given query operation.

7.3. Parameter configurations

We evaluate the performance of DMatrix and TCM in terms of accuracy on the three datasets under the conditions of *same number of buckets* and *same memory overhead*, respectively.

In the experiments with same number of buckets, we set the same width/length h and depth w for both DMatrix and TCM. We evaluated the accuracy performance with varying compression rates and depths w (i.e., number of hash functions), respectively. The compression rate is defined as the ratio of number of buckets in one layer (i.e., $h \times h$) to the number of edges in the stream dataset. For example, if we set $h = 246$, the compression rate is $c = (246 \times 246)/42\,218\,852 \approx 1/700$, where 42 218 852 is the total number of edges in IP-trace stream. In the condition of fixed compression rate with varying w, we set h to 246, 388, and 865 for the IP-trace, Twitter-communication, and DBLP datasets such that the compression rates are 1/700, 1/160, and 1/40, respectively. In the condition of fixed w with varying compression rates, we set the number of hash functions to 3. In addition, we set the heavy-hitter threshold $\alpha = 0.02$, the heavy-changer threshold $\beta = 0.03$ and the path weight threshold $\gamma = 0.01$ for IP-trace dataset; $\alpha = 0.002$, $\beta = 0.0026$, $\gamma = 0.001$ for Twitter-communication dataset; and $\alpha = 8 \times 10^{-6}$, $\beta = 1 \times 10^{-5}$, $\gamma = 1 \times 10^{-6}$ for DBLP dataset.

While for performance comparison with same memory overhead, We set the number of buckets in TCM to three times the number of that in DMatrix as the size of a bucket in DMatrix is three times that of TCM. Thus we simply expanded the length/width of TCM h_{TCM} to $\sqrt{3}$ times that of DMatrix $h_{DMatrix}$, i.e., $h_{DMatrix}$ is set to 246, 388, and 865 while h_{TCM} takes the values of 427, 673, and 1499 for IP-trace, Twitter-communication, and DBLP dataset, respectively. And the thresholds of heavy-hitter, heavy-changer, and path weight are the same as the above.

7.4. Experimental results

The following experimental results illustrate the comparison between DMatrix and TCM in terms of accuracy and efficiency.

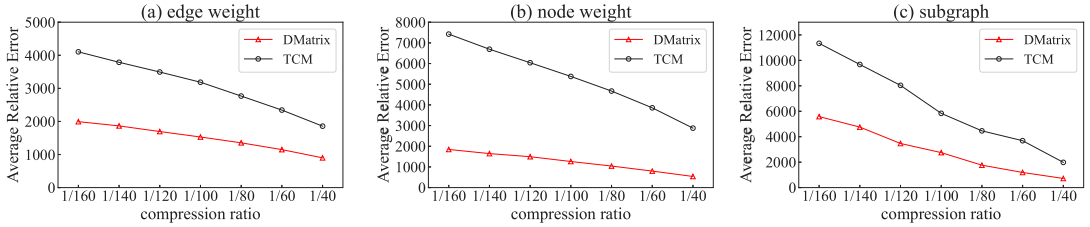

Fig. 7. Average relative error of weight-based queries with fixed $w = 3$ and varying compression rates on Twitter-communication dataset.

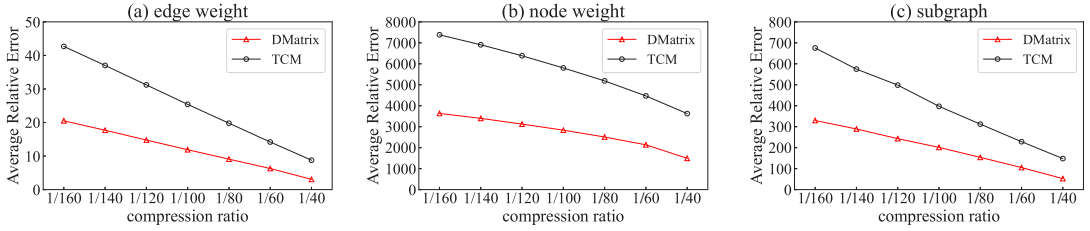

Fig. 8. Average relative error of weight-based queries with fixed $w = 3$ and varying compression rates on DBLP dataset.

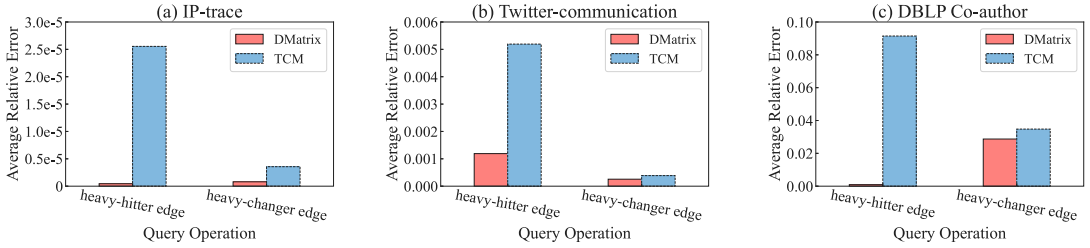

Fig. 9. Average relative error of heavy-key edge queries on DMatrix and TCM with same number of buckets.

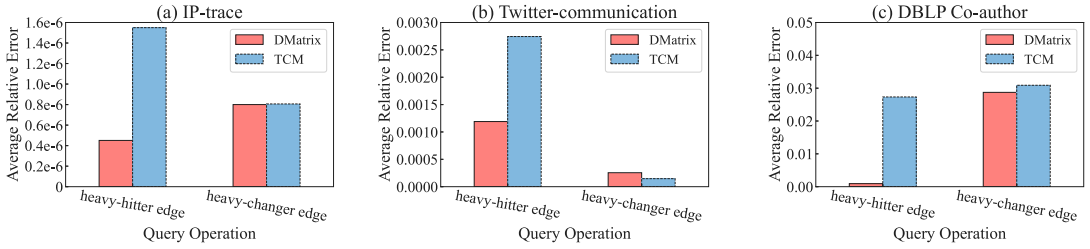

Fig. 10. Average relative error of heavy-key edge queries on DMatrix and TCM with same memory overhead.

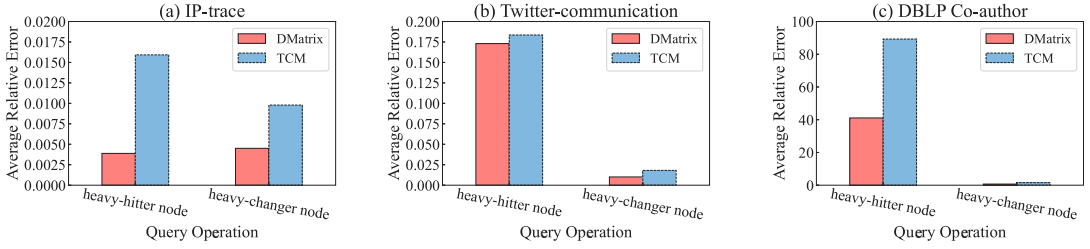

Fig. 11. Average relative error of heavy-key node queries on DMatrix and TCM with same number of buckets.

7.4.1. Accuracy of weight-based query

We compare the average relative error of each edge/node weight received in an epoch. For the subgraph weight query, we randomly choose 10 nodes received in an epoch and query the sum of the weight between them.

First, we evaluate the accuracy performance with a fixed compression rate (i.e., 1/700, 1/160, and 1/40 for the three datasets, respectively) and varying w (i.e., number of hash functions). Figs. 3–5 show the average relative error of weight-based queries on the three datasets with the same number of buckets in both DMatrix and TCM. We can see that regardless of the number of depths w in DMatrix/TCM,

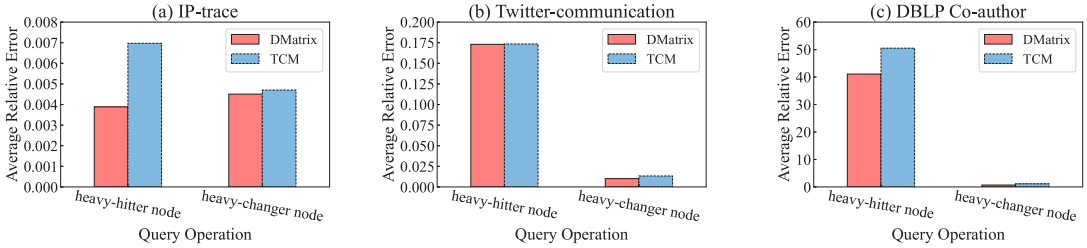

Fig. 12. Average relative error of heavy-key node queries on DMatrix and TCM with same memory overhead.

the average relative error of weight-based query on DMatrix is much better than that on TCM with the same number of depths. And this is consistent with our theoretical analysis in Table 2. As the number of hash functions w increases, the average relative error of DMatrix and TCM gradually decreases, which is an expected result.

Then, we evaluate the accuracy performance with fixed w (i.e., number of hash functions/number of depths) and varying compression rates for DMatrix and TCM. Figs. 6, 7, and 8 show the experimental results on the three datasets. Obviously, the average relative error decreases with the increase of the compression rate in each dataset. The average estimation error of DMatrix is much lower than that of TCM, especially in the case of a low compression rate (i.e., more edges are hashed in each bucket on average). Note that the average estimation error of each weight-based query is still very high. This is because the compression of data makes many collisions and the skew characteristics of streams will expand the estimation error.

However, the relative error of heavy-key query, as shown in Figs. 9, 10, 11, and 12, is very small due to the collisions make less damage to the weight estimation of heavy-key in the stream. Note that the average relative error of DMatrix in almost all heavy-key query operations is lower than that of TCM, in both the conditions of same number of buckets and same memory overhead. Although TCM has more buckets under the condition of same memory overhead with DMatrix, its performance is still slightly inferior to DMatrix.

7.4.2. Accuracy of heavy-key and reachability query

We compare the average precision, recall and F1-score of heavy-key and reachability queries. For the path reachability query, we randomly choose 10 nodes received in this epoch and query the reachability between each of them, thus 45 times of reachability query in every epoch.

Figs. 13 and 14 illustrate the average precision of heavy-key and reachability queries on DMatrix and TCM under the conditions of same number of buckets and same memory overhead, respectively. The average precision of DMatrix is higher or equal to that of TCM in all queries on datasets of IP-trace and Twitter-communication. In particular, on queries of heavy-hitter node and heavy-changer node, the precision of TCM declines significantly, since its relatively large estimation error on node weight.

Note that almost all heavy-key queries show a very low precision on the DBLP dataset, whether DMatrix or TCM. We analyze why the sketch structure is not suitable for this dataset. Fig. 15(a), (b), and (c) show the edge weight distribution for IP-trace, Twitter-communication, and DBLP dataset, respectively. For each dataset, the x-label indicates the range of edge weights. For instance, the edge weights in [1, 143] are the number of co-authorships between two authors for DBLP in Fig. 15(c). The y-axis indicates the number that such frequency appears in the dataset and the red vertical line indicates the weight threshold of the top-5 heavy-hitter edges. Obviously, the edge frequencies of all datasets satisfy the Pareto distribution. We also note that the heavy-hitter edge weight of the IP-trace and Twitter-communication is extremely large, reaching the order of 10^8, while the heavy-hitter edge weight of the DBLP is relatively small, with a maximum edge

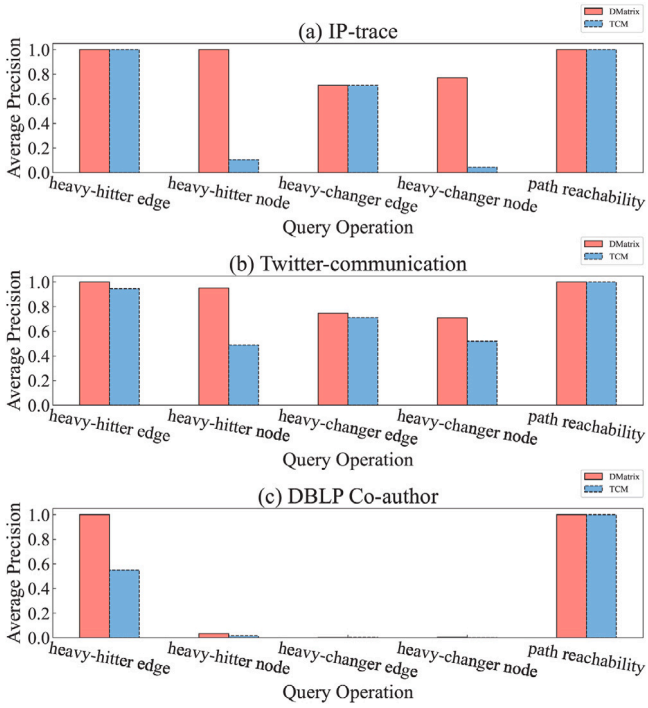

Fig. 13. Average precision of heavy-key and reachability queries on DMatrix and TCM with same number of buckets.

weight of 143. According to the weight threshold of the top-5 heavy-hitter edges and the total sum weight in the dataset, we can derive a heavy hitter threshold α. For example, the total edge weight of IP-trace is 7 195 409 970 and the weight of 144 698 912 is the lower bound of the top-5 heavy-hitter edges. Then we derive the heavy-hitter threshold $\alpha \leq 144\,698\,912/7\,195\,409\,970 \approx 0.0201$ such that the heavy-hitter edges/nodes can be queried by DMatrix with a high probability. Similarly, the heavy-hitter thresholds of Twitter-communication and DBLP are $\alpha \leq 0.00378$ and $\alpha \leq 8.14 \times 10^{-6}$, respectively. According to Theorem 4, given $\alpha \geq \sqrt{2\epsilon}$, DMatrix returns every heavy-hitter node with a probability of at least $1 - \delta$. Then we have $h = \sqrt{\frac{2}{\epsilon}} \geq \frac{2}{\alpha}$. For the DBLP dataset, we should set $h_{DBLP} \geq 2.45 \times 10^5$ to ensure a high accuracy probability yet we only set $h_{DBLP} = 865$ to make the compression rate of 1/40. Therefore, the high accuracy of heavy-key queries cannot be guaranteed at this compression rate. As for the IP-trace dataset, the length of the sketch should be set to $h_{IP-trace} > 99$, and we set it to 246 which makes the compression rate of 1/700 while ensuring a high accuracy as shown in Figs. 13 and 14.

Figs. 16 and 17 illustrate the average recall of heavy-key and reachability queries on DMatrix and TCM in the conditions of same number of buckets and same memory overhead, respectively. We can see that the average recall of DMatrix is lower or equal to that of TCM in all queries on both datasets since DMatrix only records a part of the key

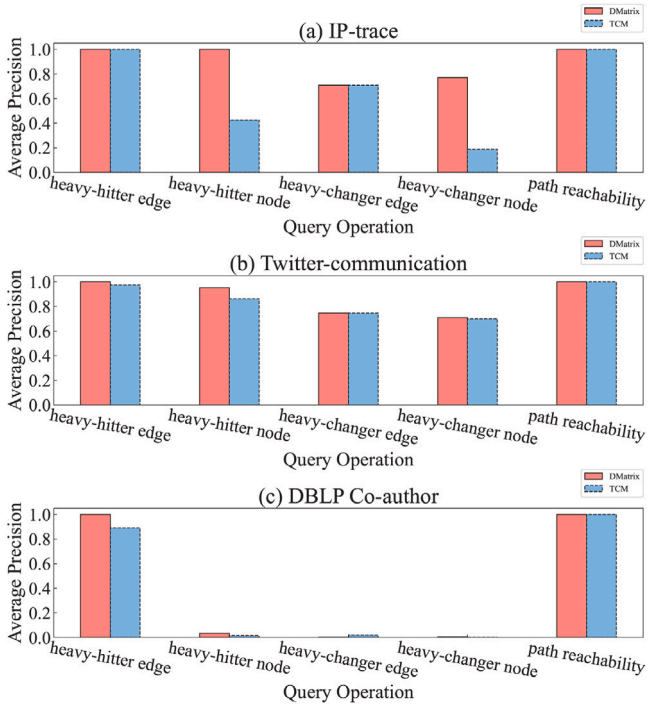

Fig. 14. Average precision of heavy-key and reachability queries on DMatrix and TCM with same memory overhead.

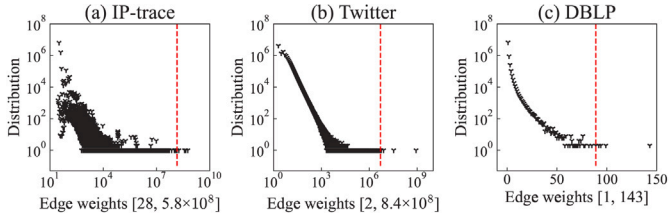

Fig. 15. Edge weight distribution.

space in its bucket. In short, a heavy-key is not stored in DMatrix's key field will result in underreporting while TCM traverses the entire key space to conduct these queries, thus ensuring a high recall. However, the time cost of heavy-key query on TCM is orders of magnitude higher than that of DMatrix as shown in Table 3. Since TCM has to query the whole key space instead of querying the most likely keys as DMatrix does.

Figs. 18 and 19 illustrate the F1-score of heavy-key and reachability queries on DMatrix and TCM in the three datasets. Note that DMatrix performs far better than TCM on F1-score in the case of same number of buckets. That is to say, with an acceptable increase in memory overhead, DMatrix performs better than TCM in terms of accuracy and efficiency. While under the condition of same memory overhead, they achieve almost the same performance on F1-score, while DMatrix is more efficient due to reversible queries.

7.4.3. Update and query efficiency

Table 3 shows the average time cost of updating and querying process in DMatrix and TCM in the case of same number of buckets. We note that DMatrix has a higher time cost in updating the sketch. This is because DMatrix has additional operations on key comparison and updating indicate field when compare to TCM which only needs to update the counter field. Also, the same reason is for the slightly higher time consumption on weight-based queries in DMatrix. However, the query efficiency on heavy-key of DMatrix is several orders of magnitude

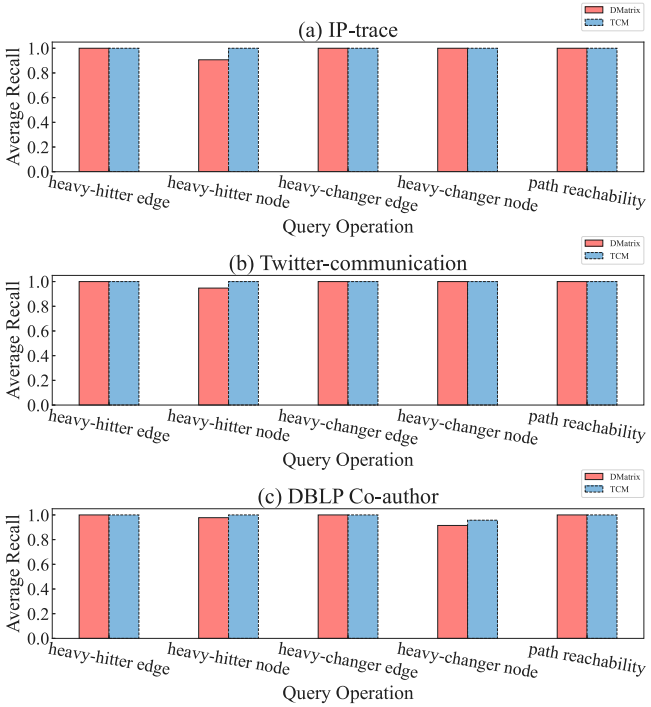

Fig. 16. Average recall of heavy-key and reachability queries on DMatrix and TCM with same number of buckets.

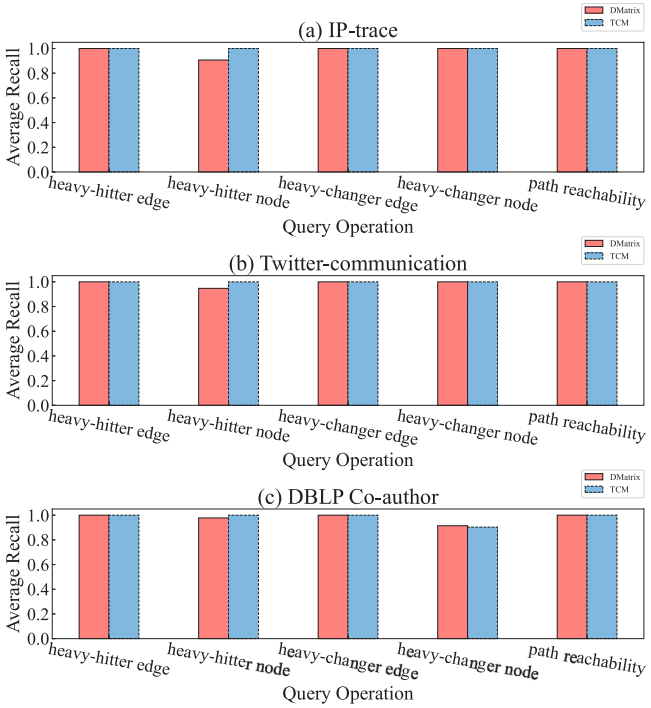

Fig. 17. Average recall of heavy-key and reachability queries on DMatrix and TCM with same memory overhead.

higher than that TCM due to its partly reversible properties. It is also noted that the time cost of path reachability query in both sketches is much higher than other queries. This is because the BFS algorithm they used is very time-consuming.

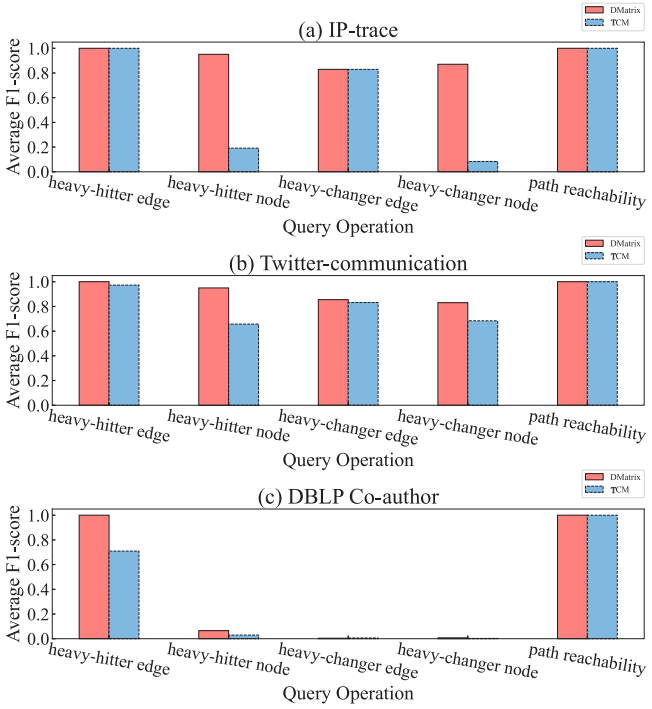

Fig. 18. Average F1-score of heavy-key and reachability queries on DMatrix and TCM with same number of buckets.

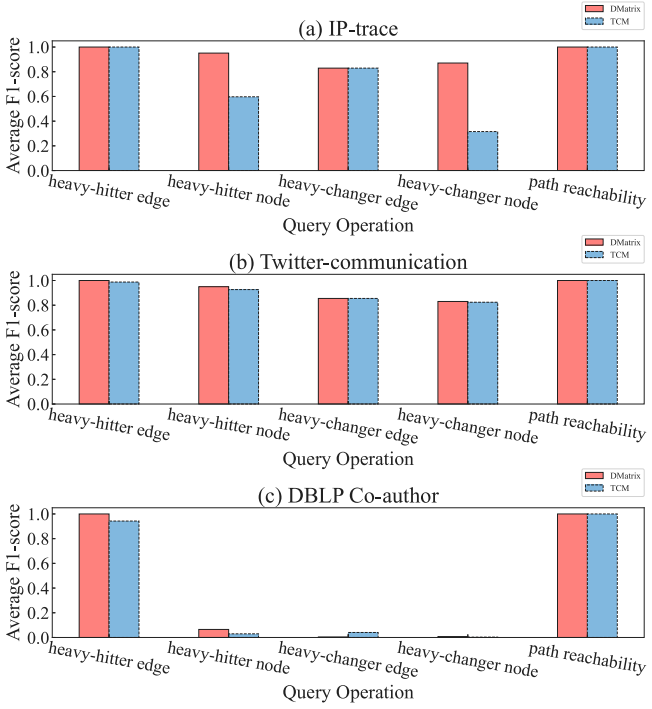

Fig. 19. Average F1-score of heavy-key and reachability queries on DMatrix and TCM with same memory overhead.

8. Conclusion

We proposed a graph sketch, namely DMatrix, for summary and analysis graph stream in a sublinear space, using one-pass updating, and with constant probabilistic error bound guarantee. A key property of DMatrix is that its data structure maintains both the structure- and weight-based information over the graph stream. This is achieved by utilizing a 3-dimensional storage structure, which stores the adjacency of different nodes. We apply the MJRTY algorithm to narrow the error bounds and achieve reversibility to improve query efficiency. Both theoretical analysis and experimental results demonstrated that DMatrix is more accurate and efficient than existing solutions.

Table 3
The average time cost of updating and querying process in DMatrix and TCM.

Process	Sketch	Average time cost (s)	
		IP-trace	Twitter communication
Updating sketch	DMatrix	2.61	1.12
	TCM	1.80	0.69
Edge weight query	DMatrix	3.20e−7	2.62e−7
	TCM	2.93e−7	2.39e−7
Node weight query	DMatrix	2.91e−6	2.60e−6
	TCM	1.17e−6	8.64e−7
Heavy-hitter edge query	DMatrix	4.81e−4	4.49e−4
	TCM	2.67	1.11
Heavy-hitter node query	DMatrix	8.89e−3	9.28e−3
	TCM	1.38	0.18
Heavy-changer edge query	DMatrix	9.38e−4	9.53e−4
	TCM	8.69	3.04
Heavy-changer node query	DMatrix	0.22	0.23
	TCM	9.92	0.55
Subgraph weight query (10 nodes)	DMatrix	74.66	19.24
	TCM	74.43	19.02
Path reachability query	DMatrix	91.31	23.93
	TCM	92.28	23.68

CRediT authorship contribution statement

Changsheng Hou: Software, Validation, Formal analysis, Writing – original draft, Writing – review & editing. **Bingnan Hou:** Conceptualization, Methodology, Formal analysis, Writing – original draft, Writing – review & editing. **Tongqing Zhou:** Writing – original draft, Project administration, Supervision. **Zhiping Cai:** Supervision, Resources, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work is supported by the National Key Research and Development Program of China (2018YFB1800202).

Appendix A. Notations

The summary of notations are shown in Table A.4.

Appendix B. Error boundary on dmatrix

The following shows the proofs of error boundary on DMatrix in various query tasks.

Table A.4
Summary of notations.

Notation	Description
$G = \langle e_1, e_2, \dots, e_m \rangle$	Graph stream
$e = (x, y, f; t)$	Graph stream element, where f is the weight/frequency on the edge (x, y), and t is the timestamp
h, w	Length (width) and depth of DMatrix
$h_k(\cdot)$	Hash function
$B(i, j, k)$	Bucket index
$S(i, j, k)$	Sum in $B(i, j, k)$
$K(i, j, k)$	Key in $B(i, j, k)$
$I(i, j, k)$	Indicator counter of $B(i, j, k)$
ϵ, δ	Approximation parameter and error probability
$\omega(x, y), \tilde{\omega}(x, y)$	Weight and estimated weight of edge (x, y)
$U(x, y), L(x, y)$	Upper and lower bounds of $\tilde{\omega}(x, y)$
$\omega(x), \tilde{\omega}(x)$	Weight and estimated weight of node x
$U(x), L(x)$	Upper and lower bounds of $\tilde{\omega}_{out}(x)$
$\mathcal{F}^t, \mathcal{F}^{t-1}$	Total edge weight in the current and previous epochs
$\mathcal{E}_\alpha, \mathcal{V}_\alpha, \mathcal{E}_\beta, \mathcal{V}_\beta$	Detected heavy-hitter edges, heavy-hitter nodes, heavy-changer edges and heavy-changer nodes

B.1. Node weight query of dmatrix (proof of Theorem 2)

Proof. From Algorithm 3, we can easily get that $\tilde{\omega}(x) \geq \omega(x)$. Next, we states the upper bound of $\tilde{\omega}(x)$ in terms of ϵ and δ. Consider the expectation of the total sum of all weight except x in its mapped row, i.e., bucket array $B(i, j, k)$, $j \in [1, h]$. It is given by $E[\sum_{j=1}^{h} S(i, j, k) - \omega(x)] = E[\sum_{s \neq x, h_k(s)=h_k(x)} \omega(s)] \leq \frac{\mathcal{F}-\omega(x)}{h} \leq \frac{\sqrt{2\epsilon}\mathcal{F}}{2}$. By Markov's inequality, we have

$$Pr[\sum_{j=1}^{h} S(i, j, k) - \omega(x) \geq \sqrt{2\epsilon}\mathcal{F}] \leq \frac{1}{2}. \tag{B.1}$$

For each bucket in the ith row and kth depth: if $K(i, j, k) == (x, \cdot)$, then $\tilde{\omega}(x)_{j,k} - \omega(x)_{j,k} = \frac{S(i,j,k)+I(i,j,k)}{2} - \omega(x)_{j,k} \leq \frac{S(i,j,k)-\omega(x)_{j,k}}{2}$, where i, k are fixed values, $j \in [1, h]$ and $\omega(x)_{j,k}$ $(\omega(x)_{j,k} \geq 0)$ denotes the real outgoing weight components of node x in the jth column and kth depth; if $K(i, j, k) \neq (x, \cdot)$, then $\tilde{\omega}(x)_{j,k} - \omega(x)_{j,k} = \frac{S(i,j,k)-I(i,j,k)}{2} - \omega(x)_{j,k} \leq \frac{S(i,j,k)-\omega(x)_{j,k}}{2}$.

Considering the row estimate error in the kth depth of bucket array, we have $\tilde{\omega}(x)_k - \omega(x) = \sum_{j=1}^{h}(\tilde{\omega}(x)_{j,k} - \omega(x)_{j,k}) \leq \frac{\sum_{j=1}^{h} S(i,j,k)-\omega(x)}{2}$, where $\omega(x)_k = \omega(x)$. Then $Pr[\tilde{\omega}(x)_k - \omega(x) \geq \frac{\sqrt{2\epsilon}\mathcal{F}}{2}] \leq Pr[\frac{\sum_{j=1}^{h} S(i,j,k)-\omega(x)}{2} \geq \frac{\sqrt{2\epsilon}\mathcal{F}}{2}] \leq \frac{1}{2}$ due to (B.1).

Since $\tilde{\omega}(x)$ is the minimum of all $\tilde{\omega}(x)_k$ in w different buckets, we have $Pr[\tilde{\omega}(x) \leq \omega(x) + \frac{\sqrt{2\epsilon}\mathcal{F}}{2}] = 1 - Pr[\tilde{\omega}(x) \geq \omega(x) + \frac{\sqrt{2\epsilon}\mathcal{F}}{2}] = 1 - Pr[\tilde{\omega}(x)_k \geq \omega(x) + \frac{\sqrt{2\epsilon}\mathcal{F}}{2}, \forall k] \geq 1 - (\frac{1}{2})^w = 1 - \delta$. □

B.2. Heavy-hitter edge query of DMatrix (proof of Theorem 3)

Proof. For a real heavy-hitter edge, (x, y) has the majority weight in at least one of its hashed buckets, say $B(i, j, k)$, where the majority weight in one bucket means $\omega(x, y) > \frac{1}{2}S(i, j, k)$. Then, (x, y) will occupy the key field $K(i, j, k)$ of this bucket. Thus DMatrix will report (x, y) due to Algorithm 4.

Only when (x, y) is not the majority weight for all w hashed buckets, i.e., $\omega(x, y) \leq \frac{S(i,j,k)}{2}$, DMatrix does not report (x, y). The probability that it occurs is $Pr[\omega(x, y) \leq \frac{S(i,j,k)}{2}, \forall k] = Pr[S(i, j, k) - \omega(x, y) \geq \omega(x, y), \forall k]$. Given that $\alpha \geq \epsilon$, we have $\omega(x, y) \geq \alpha\mathcal{F} \geq \epsilon\mathcal{F}$ which gives $Pr[S(i, j, k) - \omega(x, y) \geq \epsilon\mathcal{F}, \forall k] \leq (\frac{1}{2})^w = \delta$ due to (1). Therefore, assuming that $\alpha \geq \epsilon$, the probability of reporting a heavy-hitter edge is at least $1 - \delta$.

For a non-heavy-hitter edge (s, d) with $\omega(s, d) \leq (\alpha - \frac{\epsilon}{2})\mathcal{F}$, the condition is that $\tilde{\omega}(s, d) \geq \alpha\mathcal{F}$, thus $\tilde{\omega}(s, d) - \omega(s, d) \geq \alpha\mathcal{F} - (\alpha - \frac{\epsilon}{2})\mathcal{F} = \frac{\epsilon\mathcal{F}}{2}$. From Theorem 1, we have $Pr[\tilde{\omega}(s, d) - \omega(s, d) \geq \frac{\epsilon\mathcal{F}}{2}] \leq \delta$. In other words, the probability of reporting (s, d) with a upper weight bound of $(\alpha - \frac{\epsilon}{2})\mathcal{F}$ as a heavy-hitter edge is at most δ. □

B.3. Heavy-hitter node query of DMatrix (proof of Theorem 4)

Proof. We first prove that the probability of reporting a real heavy-hitter node, say x. If $(x, \cdot)$ has the majority weight in any of its hashed buckets, say $B(i, j, k)$, where the majority weight in one bucket means $\omega(x, \cdot) > \frac{1}{2}S(i, j, k)$. Then, $(x, \cdot)$ will occupy the key field $K(i, j, k)$ of this bucket. Thus DMatrix will report x due to Algorithm 5.

Dmatrix fails to report x only if $(x, \cdot)$ is not the majority weight in any of its hashed buckets, i.e., $\omega(x, \cdot) \leq \frac{S(i,j,k)}{2}$. The probability that it occurs is $Pr[\omega(x)_{j,k} \leq \frac{S(i,j,k)}{2}, \forall j, k] = Pr[S(i, j, k) - \omega(x)_{j,k} \geq \omega(x)_{j,k}, \forall j, k] \leq Pr[\sum_{j=1}^{h} S(i, j, k) - \omega(x) \geq \omega(x), \forall k]$. Given that $\alpha \geq \sqrt{2\epsilon}$, we have $\omega(x) \geq \alpha\mathcal{F} \geq \sqrt{2\epsilon}\mathcal{F}$ which gives $Pr[\sum_{j=1}^{h} S(i, j, k) - \omega(x) \geq \sqrt{2\epsilon}\mathcal{F}, \forall k] \leq (\frac{1}{2})^w = \delta$ due to (B.1). Thus, a heavy hitter node is reported with a probability at least $1 - \delta$ assuming that $\alpha \geq \sqrt{2\epsilon}$.

We next prove the probability of reporting a non-heavy-hitter node, say s, with $\omega(s) \leq (\alpha - \frac{\sqrt{2\epsilon}}{2})\mathcal{F}$. The condition is that $\tilde{\omega}(s) \geq \alpha\mathcal{F}$, thus $\tilde{\omega}(s) - \omega(s) \geq \alpha\mathcal{F} - (\alpha - \frac{\sqrt{2\epsilon}}{2})\mathcal{F} = \frac{\sqrt{2\epsilon}\mathcal{F}}{2}$. From Theorem 2, we have $Pr[\tilde{\omega}(s) - \omega(s) \geq \frac{\sqrt{2\epsilon}\mathcal{F}}{2}] \leq \delta$. In other words, s with a upper outgoing/incoming weight bound of $(\alpha - \frac{\sqrt{2\epsilon}}{2})\mathcal{F}$ is reported as a heavy-hitter node with a probability at most δ. □

B.4. Heavy-changer edge query of DMatrix (proof of Theorem 5)

Recall that the estimated weight change $\tilde{D}(x, y)$ relies on the upper bound $U(x, y)$ and lower bound $L(x, y)$ of $\omega(x, y)$, we first give the error bounds of $U(x, y)$ and $L(x, y)$.

Lemma 1. *$\omega(x, y) \leq U(x, y) \leq \omega(x, y) + \frac{\epsilon\mathcal{F}}{2}$ with a probability at least $1 - \delta$, where $\mathcal{F}$ is the total edge weight.*

Proof. Algorithm 6 gives that $U(x, y) = \tilde{\omega}(x, y)$ and Theorem 1 shows the error bounds for $\tilde{\omega}(x, y)$. Thus we have $\omega(x, y) \leq U(x, y) \leq \omega(x, y) + \frac{\epsilon\mathcal{F}}{2}$ with a probability at least $1 - \delta$ □

Lemma 2. *$\omega(x, y) - \epsilon\mathcal{F} \leq L(x, y) \leq \omega(x, y)$ with a probability at least $1 - \delta$, where $\mathcal{F}$ is the total edge weight.*

Proof. From Algorithm 2, we can easily get that $L(x, y) \leq \omega(x, y)$. Next, we states the lower bound of $L(x, y)$ in terms of ϵ and δ. Considering inside one bucket $B(i, j, k)$, if (x, y) is the majority weight (i.e., $K(i, j, k) == (x, y)$), we have $L(x, y)_k = I(i, j, k)$, where $k \in [1, w]$. Theorem 1 gives that $\omega(x, y) \leq \tilde{\omega}(x, y) = \frac{S(i,j,k)+I(i,j,k)}{2}$, which implies $\omega(x, y) - L_k(x, y) \leq S(i, j, k) - \omega(x, y)$.

If (x, y) is not the majority weight in $B(i, j, k)$ (i.e., $K(i, j, k) \neq (x, y)$), we have $L(x, y)_k = 0$. Then $\omega(x, y) - L(x, y)_k = \omega(x, y) \leq S(i, j, k) - \omega(x, y)$.

Combining both cases, we have $Pr[\omega(x, y) - L(x, y) \geq \epsilon\mathcal{F}] = Pr[\omega(x, y) - L(x, y)_k \geq \epsilon\mathcal{F}, \forall k] \leq Pr[S(i, j, k) - \omega(x, y) \geq \epsilon\mathcal{F}, \forall k] \leq (\frac{1}{2})^w = \delta$ due to (1). Thus, we proved that the lower bound of $L(x, y)$ is $\omega(x, y) - \epsilon\mathcal{F}$ with a probability at least $1 - \delta$. □

From Algorithm 6, we have $\tilde{D}(x, y) \leftarrow max\{|U^t(x, y) - L^{t-1}(x, y)|, |L^t(x, y) - U^{t-1}(x, y)|\}$. Now we provide the error bounds of $\tilde{D}(x, y)$.

Lemma 3. *$D(x, y) \leq \tilde{D}(x, y) \leq D(x, y) + \epsilon(\mathcal{F}^t + \mathcal{F}^{t-1})$ with a probability at least $(1 - \delta)^2$, where $\mathcal{F}^t$ and $\mathcal{F}^{t-1}$ denote the total sum of all weight in the current and previous epochs, respectively.*

Proof. Both $U(x, y)$ and $L(x, y)$ are the true upper and lower bounds of $D(x, y)$, thus the lower bound of $\tilde{D}(x, y)$ is $D(x, y)$.

From Lemmas 1 and 2, we have $|U^t(x, y) - L^{t-1}(x, y)| \leq |\omega^t(x, y) + \frac{\epsilon\mathcal{F}^t}{2} - (\omega^{t-1}(x, y) - \epsilon\mathcal{F}^{t-1})| = D(x, y) + \epsilon(\frac{\mathcal{F}^t}{2} + \mathcal{F}^{t-1}) \leq D(x, y) + \epsilon(\mathcal{F}^t + \mathcal{F}^{t-1})$ with a probability of $(1-\delta)^2$ since $U^t(x, y)$ and $L^{t-1}(x, y)$ are independent. Similarly, we have $|U^{t-1}(x, y) - L^t(x, y)| \leq D(x, y) + \epsilon(\mathcal{F}^t + \mathcal{F}^{t-1})$ with a probability at least $(1 - \delta)^2$. As $\tilde{D}(x, y) \leftarrow$

$max\{|U^t(x,y) - L^{t-1}(x,y)|, |L^t(x,y) - U^{t-1}(x,y)|\}$, the upper bound of $\tilde{D}(x,y)$ proved. □

The following gives the proof of Theorem 5.

Proof. For a real heavy-changer edge, (x,y) has the majority weight in at least one of its hashed buckets. Due to $\tilde{D}(x,y) \geq D(x,y) \geq \beta\tilde{D}$, edge (x,y) must be reported. The unexpected situation is that (x,y) is not stored in any key fields at both the previous and current bucket array. However, it meets the condition that $\omega^t(x,y) \geq \beta\tilde{D}$ or $\omega^{t-1}(x,y) \geq \beta\tilde{D}$ as (x,y) is a real heavy-changer. In this case, (x,y) will not be reported. From Theorem 3, we can conclude that (x,y) is not reported with a probability at most δ assuming that $\frac{\beta\tilde{D}}{\epsilon} \geq max\{\mathcal{F}^t, \mathcal{F}^{t-1}\}$ and $\beta \geq \epsilon$.

For a non-heavy changer edge (s,d) with $D(s,d) \leq \beta\tilde{D} - \epsilon(\mathcal{F}^t + \mathcal{F}^{t-1})$, the falsely returned (s,d) satisfies $D(s,d) \leq \beta\tilde{D} \leq \tilde{D}(s,d)$. Assuming that $D(s,d) \leq \beta\tilde{D} - \epsilon(\mathcal{F}^t + \mathcal{F}^{t-1})$, then $\beta\tilde{D} - D(s,d) \geq \epsilon(\mathcal{F}^t + \mathcal{F}^{t-1})$. From Lemma 3, we have $D(s,d) \leq \tilde{D}(s,d) \leq D(s,d) + \epsilon(\mathcal{F}^t + \mathcal{F}^{t-1})$ with a probability at least $(1-\delta)^2$. In other words, the probability of $\tilde{D}(s,d) - D(s,d) \geq \epsilon(\mathcal{F}^t + \mathcal{F}^{t-1})$ is at most $1-(1-\delta)^2$, that is, the probability of falsely reporting a non-heavy-changer edge is at most $1-(1-\delta)^2$. □

B.5. Heavy-changer node query of DMatrix (proof of Theorem 6)

Recall that the estimate of weight change $\tilde{D}(x)$ relies on the upper bound $U(x)$ and lower bound $L(x)$ of $\omega(x)$, we first give the error bounds of $U(x)$ and $L(x)$.

Lemma 4. *$\omega(x) \leq U(x) \leq \omega(x) + \frac{\sqrt{2\epsilon}\mathcal{F}}{2}$ with a probability at least $1-\delta$, where $\mathcal{F}$ is the total weight received so far.*

Proof. Algorithm 7 gives that $U(x) = \tilde{\omega}(x)$ and Theorem 2 shows the error bounds for $\tilde{\omega}(x)$. Thus we have $\omega(x) \leq U(x) \leq \omega(x) + \frac{\sqrt{2\epsilon}\mathcal{F}}{2}$ with a probability at least $1-\delta$ □

Lemma 5. *$\omega(x) - \sqrt{2\epsilon}\mathcal{F} \leq L(x) \leq \omega(x)$ with a probability at least $1-\delta$, where $\mathcal{F}$ is the total received so far.*

Proof. From Algorithm 3, we can easily get that $L(x) \leq \omega(x)$. Next, we states the lower bound of $L(x)$ in terms of ϵ and δ. Considering the mapped row buckets $B(i,j,k)$ of node x, where i and k are fixed and $j \in [1,h]$. For every bucket in this row, if $K(i,j,k) == (x,\cdot)$, then $L(x)_{j,k} = I(i,j,k)$; otherwise (i.e., $K(i,j,k) \neq (x,\cdot)$) $L(x)_{j,k} = 0$ where $L(x)_{j,k}$ denotes the lower estimation of node x's outgoing/incoming weight component in the j-column and kth depth of the bucket array.

From Theorem 2, we have $\omega(x)_{j,k} \leq \tilde{\omega}(x)_{j,k} = \frac{S(i,j,k)+I(i,j,k)}{2}$ in the case of $(x,\cdot) == K(i,j,k)$, which implies $\omega(x)_{j,k} - L(x)_{j,k} \leq S(i,j,k) - \omega(x)_{j,k}$; otherwise (i.e., $(x,\cdot) \neq K(i,j,k)$), $\omega(x)_{j,k} - L(x)_{j,k} = \omega(x)_{j,k} \leq S(i,j,k) - \omega(x)_{j,k}$.

Combining both cases and summing the row up, we have $\omega(x) - L(x)_k \leq \sum_{j=1}^{h} S(i,j,k) - \omega(x)$. Then $Pr[\omega(x) - L(x) \geq \epsilon\mathcal{F}] = Pr[\omega(x) - L(x)_k \geq \sqrt{2\epsilon}\mathcal{F}, \forall k] \leq Pr[\sum_{j=1}^{h} S(i,j,k) - \omega(x,y) \geq \sqrt{2\epsilon}\mathcal{F}, \forall k] \leq (\frac{1}{2})^w = \delta$ due to (B.1). Thus, we proved that the lower bound of $L(x)$ is $\omega(x) - \sqrt{2\epsilon}\mathcal{F}$ with a probability at least $1-\delta$. □

From Algorithm 7, we have $\tilde{D}(x) \leftarrow max\{|U^t(x) - L^{t-1}(x)|, |L^t(x) - U^{t-1}(x)|\}$. Now we provide the error bounds of $\tilde{D}(x)$.

Lemma 6. *$D(x) \leq \tilde{D}(x) \leq D(x) + \sqrt{2\epsilon}(\mathcal{F}^t + \mathcal{F}^{t-1})$ with a probability at least $(1-\delta)^2$, where $\mathcal{F}^t$ and $\mathcal{F}^{t-1}$ denote the total sum of all weight in the current and previous epochs, respectively.*

Proof. Both $U(x)$ and $L(x)$ are the true upper and lower bounds of $D(x)$, thus the lower bound of $\tilde{D}(x)$ is $D(x)$.

From Lemmas 4 and 5, we have $|U^t(x) - L^{t-1}(x)| \leq |\omega^t(x) + \frac{\sqrt{2\epsilon}\mathcal{F}^t}{2} - (\omega^{t-1}(x) - \sqrt{2\epsilon}\mathcal{F}^{t-1})| = D(x,y) + \sqrt{2\epsilon}(\frac{\mathcal{F}^t}{2} + \mathcal{F}^{t-1}) \leq D(x,y) + \sqrt{2\epsilon}(\mathcal{F}^t + \mathcal{F}^{t-1})$ with a probability at least $(1-\delta)^2$ since $U^t(x)$ and $L^{t-1}(x)$ are independent. Similarly, we have $|U^{t-1}(x) - L^t(x)| \leq D(x) + \sqrt{2\epsilon}(\mathcal{F}^t + \mathcal{F}^{t-1})$ with a probability at least $(1-\delta)^2$. As $\tilde{D}(x) = max\{|U^t(x) - L^{t-1}(x)|, |L^t(x) - U^{t-1}(x)|\}$, the upper bound of $\tilde{D}(x)$ proved. □

Then, we give the proof of Theorem 6.

Proof. We first prove the probability of reporting a real heavy-changer node, say x. If x has the majority weight in any one of its hashed buckets, it must be reported due to $\tilde{D}(x) \geq D(x) \geq \beta\tilde{D}$. Node x is not reported only if it is not stored in any key fields at both the previous and current bucket array. However, it meets the condition that $\omega^t(x) \geq \beta\tilde{D}$ or $\omega^{t-1}(x,y) \geq \beta\tilde{D}$ as x is a real heavy-changer. From Theorem 4, we can conclude that x is not reported with a probability at most δ assuming that $\frac{\beta\tilde{D}}{\sqrt{2\epsilon}} \geq max\{\mathcal{F}^t, \mathcal{F}^{t-1}\}$ and $\beta \geq \sqrt{2\epsilon}$.

We next prove the probability that DMatrix reports a non-heavy changer node, say s with $D(s) \leq \beta\tilde{D} - \sqrt{2\epsilon}(\mathcal{F}^t + \mathcal{F}^{t-1})$. The falsely returned s satisfies $D(s) \leq \beta\tilde{D} \leq \tilde{D}(s)$. Assuming that $D(s) \leq \beta\tilde{D} - \sqrt{2\epsilon}(\mathcal{F}^t + \mathcal{F}^{t-1})$, then $\beta\tilde{D} - D(s) \geq \sqrt{2\epsilon}(\mathcal{F}^t + \mathcal{F}^{t-1})$. From Lemma 6, we have $D(s) \leq \tilde{D}(s) \leq D(s) + \sqrt{2\epsilon}(\mathcal{F}^t + \mathcal{F}^{t-1})$ with a probability at least $(1-\delta)^2$. In other words, the probability of $\tilde{D}(s) - D(s) \geq \sqrt{2\epsilon}(\mathcal{F}^t + \mathcal{F}^{t-1})$ is at most $1-(1-\delta)^2$, that is, the probability of falsely reporting a non-heavy-changer node is at most $1-(1-\delta)^2$. □

Appendix C. Error boundary on TCM

The following shows the probabilistic accuracy guarantee and proofs of error boundary on TCM in various query tasks.

C.1. Edge weight query of TCM

Theorem 9. *Given the current total edge weight $\mathcal{F}$, the edge weight estimation of TCM satisfies $\omega(x,y) \leq \tilde{\omega}(x,y) \leq \omega(x,y) + \epsilon\mathcal{F}$ with a probability of at least $1-\delta$.*

Proof. From the edge weight query algorithm of TCM, we can easily get that $\tilde{\omega}(x,y) \geq \omega(x,y)$. Next, we show the upper bound of $\tilde{\omega}(x,y)$. As with the proof of Theorem 1, we first focus on the expectation and use Markov's inequality to derive equation (1), i.e., $Pr[S(i,j,k) - \omega(x,y) \geq \epsilon\mathcal{F}] \leq \frac{1}{2}$.

We can easily get that $\tilde{\omega}(x,y)_k - \omega(x,y) = S(i,j,k) - \omega(x,y)$. Therefore, we have $Pr[\tilde{\omega}(x,y)_k - \omega(x,y) \geq \epsilon\mathcal{F}] = Pr[S(i,j,k) - \omega(x,y) \geq \epsilon\mathcal{F}] \leq \frac{1}{2}$ due to (1).

Since $\tilde{\omega}(x,y)$ is the minimum of all $\tilde{\omega}(x,y)_k$ in w different buckets, we have $Pr[\tilde{\omega}(x,y) \leq \omega(x,y) + \epsilon\mathcal{F}] = 1 - Pr[\tilde{\omega}(x,y) \geq \omega(x,y) + \epsilon\mathcal{F}] = 1 - Pr[\tilde{\omega}(x,y)_k \geq \omega(x,y) + \epsilon\mathcal{F}, \forall k] \geq 1 - (\frac{1}{2})^w = 1 - \delta$. □

C.2. Node weight query of TCM

Theorem 10. *Given the current total edge weight $\mathcal{F}$, the node weight estimation result of TCM satisfies $\omega(x) \leq \tilde{\omega}(x) \leq \omega(x) + \sqrt{2\epsilon}\mathcal{F}$ with a probability of at least $1-\delta$.*

Proof. From the node weight query algorithm of TCM, we can easily get that $\tilde{\omega}(x) \geq \omega(x)$. Next, we states the upper bound of $\tilde{\omega}(x)$ in terms of ϵ and δ. As with the proof of Theorem 2, we consider the expectation and use Markov's inequality to derive equation (B.1), i.e., $Pr[\sum_{j=1}^{h} S(i,j,k) - \omega(x) \geq \sqrt{2\epsilon}\mathcal{F}] \leq \frac{1}{2}$.

For each bucket in the ith row and kth depth, we have $\tilde{\omega}(x)_{j,k} - \omega(x)_{j,k} = S(i,j,k) - \omega(x)_{j,k}$, where i,k are fixed values, $j \in [1,h]$ and $\omega(x)_{j,k}$ ($\omega(x)_{j,k} \geq 0$) denotes the real outgoing weight components of node x in the jth column and kth depth.

Considering the row estimate error in the kth depth of bucket array, we have $\tilde{\omega}(x)_k - \omega(x) = \sum_{j=1}^{h}(\tilde{\omega}(x)_{j,k} - \omega(x)_{j,k}) = \sum_{j=1}^{h} S(i,j,k) - \omega(x)$,

where $\omega(x)_k = \omega(x)$. Then $Pr[\tilde{\omega}(x)_k - \omega(x) \geq \sqrt{2\epsilon}\mathcal{F}] = Pr[\sum_{j=1}^{h} S(i,j,k) - \omega(x) \geq \sqrt{2\epsilon}\mathcal{F}] \leq \frac{1}{2}$ due to (B.1).

Since $\tilde{\omega}(x)$ is the minimum of all $\tilde{\omega}(x)_k$ in w different buckets, we have $Pr[\tilde{\omega}(x) \leq \omega(x) + \sqrt{2\epsilon}\mathcal{F}] = 1 - Pr[\tilde{\omega}(x) \geq \omega(x) + \sqrt{2\epsilon}\mathcal{F}] = 1 - Pr[\tilde{\omega}(x)_k \geq \omega(x) + \sqrt{2\epsilon}\mathcal{F}, \forall k] \geq 1 - (\frac{1}{2})^w = 1 - \delta$. □

C.3. Heavy-hitter edge query of TCM

Theorem 11. *The probability that TCM returns every heavy-hitter edge is 1. For a non-heavy-hitter edge whose weight is not more than $(\alpha - \epsilon)\mathcal{F}$, the probability of falsely returning it as a heavy-hitter edge is at most δ.*

Proof. Since the heavy-hitter edge query algorithm of TCM is implemented by traversing the entire key space, for a real heavy-hitter edge (x, y), TCM will definitely report it, i.e., the probability of reporting a heavy-hitter edge is 1.

For a non-heavy-hitter edge (s, d) with $\omega(s, d) \leq (\alpha - \epsilon)\mathcal{F}$, the condition is that $\tilde{\omega}(s, d) \geq \alpha\mathcal{F}$, thus $\tilde{\omega}(s, d) - \omega(s, d) \geq \alpha\mathcal{F} - (\alpha - \epsilon)\mathcal{F} = \epsilon\mathcal{F}$. According to Theorem 9, we have $Pr[\tilde{\omega}(s, d) - \omega(s, d) \geq \epsilon\mathcal{F}] \leq \delta$. In other words, the probability of reporting (s, d) with a upper weight bound of $(\alpha - \epsilon)\mathcal{F}$ as a heavy-hitter edge is at most δ. □

C.4. Heavy-hitter node query of TCM

Theorem 12. *The probability that TCM returns every heavy-hitter node is 1. For a non-heavy-hitter node whose weight is not more than $(\alpha - \sqrt{2\epsilon})\mathcal{F}$, the probability of falsely returning it as a heavy-hitter node is at most δ.*

Proof. Since the heavy-hitter node query algorithm of TCM is implemented by traversing the key space, for a real heavy-hitter node x, TCM will definitely report it, i.e., the probability of reporting a heavy-hitter node is 1.

For a non-heavy-hitter node s with $\omega(s) \leq (\alpha - \sqrt{2\epsilon})\mathcal{F}$. The condition is that $\tilde{\omega}(s) \geq \alpha\mathcal{F}$, thus $\tilde{\omega}(s) - \omega(s) \geq \alpha\mathcal{F} - (\alpha - \sqrt{2\epsilon})\mathcal{F} = \sqrt{2\epsilon}\mathcal{F}$. According to Theorem 10, we have $Pr[\tilde{\omega}(s) - \omega(s) \geq \sqrt{2\epsilon}\mathcal{F}] \leq \delta$. In other words, s with a upper outgoing/incoming weight bound of $(\alpha - \sqrt{2\epsilon})\mathcal{F}$ is reported as a heavy-hitter node with a probability at most δ. □

C.5. Heavy-changer edge query of TCM

Theorem 13. *Let $\mathcal{F}^t$ and $\mathcal{F}^{t-1}$ denote the total sum of all weight in the current and previous epochs. For a real heavy-changer edge whose weight is not less than $\beta\tilde{D} + \epsilon \cdot max\{\mathcal{F}^t, \mathcal{F}^{t-1}\}$, the probability that TCM returns every heavy-changer edge is at least $(1-\delta)^2$. For a non-heavy-changer edge whose weight is not more than $\beta\tilde{D} - \epsilon \cdot max\{\mathcal{F}^t, \mathcal{F}^{t-1}\}$, the probability of falsely returning it as a heavy-changer edge is at most $1 - (1 - \delta)^2$.*

Proof. For a real heavy-changer edge (x, y) with $D(x, y) \geq \beta\tilde{D} + \epsilon \cdot max\{\mathcal{F}^t, \mathcal{F}^{t-1}\}$, if TCM reports it correctly, then it satisfies $D(x, y) \geq \beta\tilde{D}$ and $\tilde{D}(x, y) \geq \beta\tilde{D}$. According to Theorem 9, we deduce that the probability that $\tilde{\omega}(x, y)$ satisfies $\omega(x, y) \leq \tilde{\omega}(x, y) \leq \omega(x, y) + \epsilon\mathcal{F}$ is at least $1-\delta$. From $\omega^t(x, y) \leq \tilde{\omega}^t(x, y) \leq \omega^t(x, y) + \epsilon\mathcal{F}^t$ and $\omega^{t-1}(x, y) \leq \tilde{\omega}^{t-1}(x, y) \leq \omega^{t-1}(x, y) + \epsilon\mathcal{F}^{t-1}$, we deduce that the probability that $\tilde{D}(x, y)$ satisfies $\tilde{D}(x, y) \geq D(x, y) - \epsilon \cdot max\{\mathcal{F}^t, \mathcal{F}^{t-1}\}$ is at least $(1 - \delta)^2$. Then, we have $\tilde{D}(x, y) \geq \beta\tilde{D}$ with a probability at least $(1-\delta)^2$, i.e., the probability that TCM returns every heavy-changer edge is at least $(1 - \delta)^2$.

For a non-heavy changer edge (s, d) with $D(s, d) \leq \beta\tilde{D} - \epsilon \cdot max\{\mathcal{F}^t, \mathcal{F}^{t-1}\}$, the falsely returned (s, d) satisfies $D(s, d) \leq \beta\tilde{D} \leq \tilde{D}(s, d)$. According to Theorem 9, we deduce that the probability that $\tilde{\omega}(s, d)$ satisfies $\omega(s, d) \leq \tilde{\omega}(s, d) \leq \omega(s, d) + \epsilon\mathcal{F}$ is at least $1 - \delta$. From $\omega^t(s, d) \leq \tilde{\omega}^t(s, d) \leq \omega^t(s, d) + \epsilon\mathcal{F}^t$ and $\omega^{t-1}(s, d) \leq \tilde{\omega}^{t-1}(s, d) \leq \omega^{t-1}(s, d) + \epsilon\mathcal{F}^{t-1}$, we deduce that the probability that $\tilde{D}(s, d)$ satisfies $\tilde{D}(s, d) \leq D(s, d) + \epsilon \cdot max\{\mathcal{F}^t, \mathcal{F}^{t-1}\}$ is at least $(1 - \delta)^2$. Then, we have $\tilde{D}(s, d) \leq \beta\tilde{D}$ with a probability at least $(1 - \delta)^2$. In other words, the probability of $\tilde{D}(s, d) \geq \beta\tilde{D}$ is at most $1 - (1 - \delta)^2$, that is, the probability of falsely reporting a non-heavy-changer edge is at most $1 - (1 - \delta)^2$. □

C.6. Heavy-changer node query of TCM

Theorem 14. *Let $\mathcal{F}^t$ and $\mathcal{F}^{t-1}$ denote the total sum of all weight in the current and previous epochs. For a real heavy-changer node whose weight is not less than $\beta\tilde{D} + \sqrt{2\epsilon} \cdot max\{\mathcal{F}^t, \mathcal{F}^{t-1}\}$, the probability that TCM returns every heavy-changer node is at least $(1-\delta)^2$. For a non-heavy-changer node whose weight is not more than $\beta\tilde{D} - \sqrt{2\epsilon} \cdot max\{\mathcal{F}^t, \mathcal{F}^{t-1}\}$, the probability of falsely returning it as a heavy-changer node is at most $1 - (1 - \delta)^2$.*

Proof. For a real heavy-changer node x with $D(x) \geq \beta\tilde{D} + \sqrt{2\epsilon} \cdot max\{\mathcal{F}^t, \mathcal{F}^{t-1}\}$, if TCM reports it correctly, then it satisfies $D(x) \geq \beta\tilde{D}$ and $\tilde{D}(x) \geq \beta\tilde{D}$. According to Theorem 10, we deduce that the probability that $\tilde{\omega}(x)$ satisfies $\omega(x) \leq \tilde{\omega}(x) \leq \omega(x) + \sqrt{2\epsilon}\mathcal{F}$ is at least $1 - \delta$. From $\omega^t(x) \leq \tilde{\omega}^t(x) \leq \omega^t(x) + \sqrt{2\epsilon}\mathcal{F}^t$ and $\omega^{t-1}(x) \leq \tilde{\omega}^{t-1}(x) \leq \omega^{t-1}(x) + \sqrt{2\epsilon}\mathcal{F}^{t-1}$, we deduce that the probability that $\tilde{D}(x)$ satisfies $\tilde{D}(x) \geq D(x) - \sqrt{2\epsilon} \cdot max\{\mathcal{F}^t, \mathcal{F}^{t-1}\}$ is at least $(1 - \delta)^2$. Then, we have $\tilde{D}(x) \geq \beta\tilde{D}$ with a probability at least $(1 - \delta)^2$, i.e., the probability that TCM returns every heavy-changer node is at least $(1 - \delta)^2$.

For a non-heavy changer node s with $D(s) \leq \beta\tilde{D} - \sqrt{2\epsilon} \cdot max\{\mathcal{F}^t, \mathcal{F}^{t-1}\}$, the falsely returned s satisfies $D(s) \leq \beta\tilde{D} \leq \tilde{D}(s)$. According to Theorem 10, we deduce that the probability that $\tilde{\omega}(s)$ satisfies $\omega(s) \leq \tilde{\omega}(s) \leq \omega(s) + \sqrt{2\epsilon}\mathcal{F}$ is at least $1 - \delta$. From $\omega^t(s) \leq \tilde{\omega}^t(s) \leq \omega^t(s) + \sqrt{2\epsilon}\mathcal{F}^t$ and $\omega^{t-1}(s) \leq \tilde{\omega}^{t-1}(s) \leq \omega^{t-1}(s) + \sqrt{2\epsilon}\mathcal{F}^{t-1}$, we deduce that the probability that $\tilde{D}(s)$ satisfies $\tilde{D}(s) \leq D(s) + \sqrt{2\epsilon} \cdot max\{\mathcal{F}^t, \mathcal{F}^{t-1}\}$ is at least $(1 - \delta)^2$. Then, we have $\tilde{D}(s) \leq \beta\tilde{D}$ with a probability at least $(1 - \delta)^2$. In other words, the probability of $\tilde{D}(s) \geq \beta\tilde{D}$ is at most $1 - (1 - \delta)^2$, that is, the probability of falsely reporting a non-heavy-changer node is at most $1 - (1 - \delta)^2$. □

References

[1] X. Jing, Z. Yan, X. Jiang, W. Pedrycz, Network traffic fusion and analysis against DDoS flooding attacks with a novel reversible sketch, Inf. Fusion 51 (2019) 100–113, http://dx.doi.org/10.1016/j.inffus.2018.10.013.

[2] Y. Fang, X. Huang, L. Qin, Y. Zhang, W. Zhang, R. Cheng, X. Lin, A survey of community search over big graphs, VLDB J. 29 (2019) 353–392.

[3] A. Mahanti, C. Williamson, N. Carlsson, M. Arlitt, A. Mahanti, Characterizing the file hosting ecosystem: A view from the edge, Perform. Eval. 68 (11) (2011) 1085–1102, http://dx.doi.org/10.1016/j.peva.2011.07.016.

[4] X. Gou, L. Zou, C. Zhao, T. Yang, Fast and accurate graph stream summarization, in: Proceedings of the International Conference on Data Engineering (ICDE), 2018, pp. 1118–1129, http://dx.doi.org/10.1109/ICDE.2019.00103.

[5] S. Guha, A. Mcgregor, Graph synopses, sketches, and streams: A survey, in: Proceedings of the VLDB Endowment, 2012, pp. 2030–2031.

[6] G.S. Manku, R. Motwani, Approximate frequency counts over data streams, in: Proceedings of the VLDB Endowment, 2002, pp. 346–357.

[7] Y. Zhou, Y. Zhang, C. Ma, S. Chen, O.O. Odegbile, Generalized sketch families for network traffic measurement, in: Proceedings of the ACM on Measurement and Analysis of Computing Systems, 2019, pp. 1–34, http://dx.doi.org/10.1145/3366699.

[8] R.B. Basat, G. Einziger, R. Friedman, Y. Kassner, Randomized admission policy for efficient top-k and frequency estimation, in: Proceedings of the IEEE International Conference on Computer Communications (INFOCOM), 2017, pp. 1–9.

[9] J. Gong, T. Yang, H. Zhang, H. Li, S. Uhlig, S. Chen, L. Uden, X. Li, Heavykeeper: An accurate algorithm for finding top-k elephant flows, in: Proceedings of USENIX Annual Technical Conference, 2018, pp. 909–921.

[10] R.M. Karp, S. Shenker, C.H. Papadimitriou, A simple algorithm for finding frequent elements in streams and bags, ACM Trans. Database Syst. 28 (1) (2003) 51–55.

[11] N. Tang, Q. Chen, P. Mitra, Graph stream summarization, in: Proceedings of the International Conference on Management of Data (SIGMOD), 2016, pp. 1481–1496, http://dx.doi.org/10.1145/2882903.2915223.

[12] A. Khan, C. Aggarwal, Toward query-friendly compression of rapid graph streams, Soc. Netw. Anal. Min. 7 (1) (2017) http://dx.doi.org/10.1007/s13278-017-0443-4.

[13] M. Charikar, K. Chen, M. Farach-Colton, Finding frequent items in data streams, Theoret. Comput. Sci. 312 (1) (2004) 3–15.

[14] B. Krishnamurthy, S. Sen, Y. Zhang, Y. Chen, Sketch-based change detection: Methods, evaluation, and applications, in: Proceedings of the ACM Internet Measurement Conference (IMC), 2003, pp. 234–247, http://dx.doi.org/10.1145/948205.948236.

[15] G. Cormode, S. Muthukrishnan, An improved data stream summary: The count-min sketch and its applications, J. Algorithms 55 (1) (2005) 58–75, http://dx.doi.org/10.1016/j.jalgor.2003.12.001.
[16] P. Zhao, C.C. Aggarwal, M. Wang, gSketch: On query estimation in graph streams, in: Proceedings of the VLDB Endowment, 2011, pp. 193–204, http://dx.doi.org/10.14778/2078331.2078335.
[17] R. Schweller, A. Gupta, E. Parsons, Y. Chen, Reversible sketches for efficient and accurate change detection over network data streams, in: Proceedings of the ACM Internet Measurement Conference (IMC), 2004, pp. 207–212, http://dx.doi.org/10.1145/1028788.1028814.
[18] D. Eppstein, M.T. Goodrich, Straggler identification in round-trip data streams via Newton's identities and invertible bloom filters, IEEE Trans. Knowl. Data Eng. 23 (2) (2011) 297–306.
[19] Q. Huang, P.P.C. Lee, LD-sketch: A distributed sketching design for accurate and scalable anomaly detection in network data streams, in: Proceedings of the IEEE International Conference on Computer Communications (INFOCOM), 2014, pp. 1420–1428, http://dx.doi.org/10.1109/INFOCOM.2014.6848076.
[20] L. Tang, Q. Huang, P.P.C. Lee, MV-sketch: A fast and compact invertible sketch for heavy flow detection in network data streams, in: Proceedings of the IEEE International Conference on Computer Communications (INFOCOM), 2019, pp. 2026–2034, http://dx.doi.org/10.1109/INFOCOM.2019.8737499.
[21] R.S. Boyer, J.S. Moore, MJRTY - a fast majority vote algorithm, in: Automated Reasoning, Springer, 1991, pp. 105–117.
[22] MAWI dataset, 2021, http://mawi.wide.ad.jp/mawi/.
[23] Twitter graph dataset, 2021, https://snap.stanford.edu/data/twitter7.html.
[24] N. Basher, A. Mahanti, A. Mahanti, C. Williamson, M. Arlitt, A comparative analysis of web and peer-to-peer traffic, in: Proceedings of the International Conference on World Wide Web (WWW), 2008, pp. 287–296, http://dx.doi.org/10.1145/1367497.1367537.
[25] DBLP archive, 2021, https://dblp.uni-trier.de/xml/.

Changsheng Hou received the bachelor's degree in School of Information and Communication Engineering from University of Electronic Science and Technology of China, Chengdu, China, in 2019. He is currently pursuing the master's degree with the College of Computer, National University of Defense Technology (NUDT), Changsha, China. His research interests include network measurement and network security.

Bingnan Hou received the bachelor's and master's degrees in Network Engineering from Nanjing University of Science and Technology, China, in 2010 and 2015, respectively. He is currently a Ph.D. student in College of Computer, National University of Defense Technology (NUDT), China. His research interests include network measurement and network security.

Tongqing Zhou received the bachelor's, master's, and Ph.D. degrees in Computer Science and Technology from National University of Defense Technology (NUDT), China, in 2012, 2014, and 2018, respectively. He is currently a postdoc in College of Computer, National University of Defense Technology. His main research interests include ubiquitous computing, mobile sensing, and data privacy.

Zhiping Cai received the bachelor's, master's, and Ph.D. degrees in Computer Science and Technology from National University of Defense Technology (NUDT), China, in 1996, 2002, and 2005, respectively. He is currently a full professor in College of Computer, National University of Defense Technology. His main research interests include network security and edge computing.