World Wide Web
https://doi.org/10.1007/s11280-022-01117-z

Top-k heavy weight triangles listing on graph stream

Fan Zhang[1] · Xiangyang Gou[1] · Lei Zou[1]

Received: 24 September 2022 / Revised: 16 October 2022 / Accepted: 25 October 2022

Abstract
Graph stream is used to express complex and highly dynamic relationships between entities, such as friendships in social networks. The storage and mining on graph stream are the main research areas of big data research. Triangle listing/counting is an important topic in graph mining research. The triangle, as the simplest circle and clique structure, has many applications in many real-world scenarios. A large amount of related work also exists on the study of triangles on graph streams. However, the existing research has focused on triangle counting on static graphs or graph streams, and there is a lack of research targeting heavy weight triangle listing. This paper formally defines the triangle weight on graph stream. Based on this definition, this paper presents an approximation algorithm for the top-k heavy weight triangle listing problem on graph stream, and proposes various optimized data structures DolhaT, Filtered DolhaT and Double Filtered DolhaT (DFD) to solve this problem. Experiments on real graph stream datasets demonstrate the effectiveness of the proposed optimized structures for the heavy weight triangle listing problem on graph stream.

Keywords Graph stream · Heavy Triangles · Top-*k* triangle listing

1 Introduction

Due to faster data transmission speeds brought by improved network hardware technology and the expansion of data size brought by the proliferation of network users, massive amounts of communications data are generated every day in today's Internet. A large ISP may process one billion network communication packets per router per hour [1]; the social network company Twitter handles the login and usage information of 100 million users per day [2]; and on a worldwide scale, all users send and receive 200 billion emails per day

✉ Lei Zou
zoulei@pku.edu.cn

Fan Zhang
zhangfanau@pku.edu.cn

Xiangyang Gou
gxy1995@pku.edu.cn

[1] WICT, Peking University, Zhongguancun North Street, Haidian District 100871, Beijing, China

[3]. In order to process and study this dynamic relational information that changes at high speed, it is necessary to model this information and design more efficient algorithms and data structures. Therefore data stream research, especially the study of data structures over data streams, is an important area of big data research.

Traditional data streams are modeled as a series of data item inputs that are independent of each other. A large amount of literature research has focused on the design of data structures over data streams. For example, Bloom Filter (BF) [4] is designed to check whether items appear in the data stream; Count-min Sketch (CM) [5] stores the frequency of data stream items; Spacesaving (SS) [6] and Filtered Spacesaving (FSS) [7] algorithms list the items with heavy weight in the data stream.

To deal with the data stream which involves networks, the connective relationships among the items must be considered. Graph Stream is a special kind of data stream that differs from the traditional definition of a data stream. In graph stream, the streaming items are not independent of each other but carry connection information. A graph stream is an unbounded sequence of continuous and time-evolving edges. On graph stream, a series of graph topology queries, such as reachability queries, subgraph matching queries and triangle counting, can be performed to mine the relationship information between entities. For example, in network communication data, each IP is considered as a node and the communication between two IPs is considered as an edge. The graph formed by the communication information between IPs can be used to detect the presence of network attacks by performing graph queries such as edge weights query and node degrees query [8, 9]. In social networks, each user is considered as a node and the friendships between users and the information interactions between users are considered as edges [10]. By mining information from the social network, it is possible to provide friend recommendation or content recommendation functions. In financial information, each account is considered as a node, and a transaction between two accounts is considered as an edge. If a certain online fraud is transformed into a specific graph topology [11], a graph topology query can be performed on the graph [12, 13] to quickly locate suspicious accounts and transfers.

Triangles, as an important structure on graphs, have a lot of applications in scenarios such as community discovery in social networks [14], topic discovery [15] and anomaly detection in web traffic networks [16]. Triangle Listing [17] and Triangle Counting are the very important research directions in graph algorithm research. In the field of graph stream, the main research focuses on triangle counting on graph streams [18–28]. Most triangle counting studies sample the graph stream edges and use the number of triangles produced in the sample edge set to estimate the total number of triangles in the graph stream. Triangle counting only answers how many triangles are present in the graph stream, but cannot enumerate information about these triangles. If the use case requires enumerating specific information about these triangles, this class of systems cannot provide a valid answer. Since graph streams tend to be large in size and dynamically updated at high speed, it is not realistic to enumerate all the triangles appearing in the graph stream. Therefore, in practical use cases, it is not necessary to enumerate all the triangles in the graph stream, but only enumerate those that meet specific conditions, such as the Heavy-Weight Triangle. Consider the use cases:

Use case 1: Network traffic data stream. Network traffic data stream is a typical graph stream data. Each IP address represents a node, one communication between two IPs is represented as an edge, and the size of the communication packet is represented as the weight of the edge. The structure of the graph and the weights of the edges change rapidly as packets are sent and received. The existing algorithm of heavy hitters for data stream [6, 7, 29, 30] can quickly detect suspicious communications in the graph stream and thus discover possible network attacks. However, many network attacks are characterized by patterned graph structure [31, 32], such as Information Exfiltration Attack and Distributed Denial of Service (DDoS). Traditional data stream algorithms can only enumerate heavy edges, i.e., suspicious communication patterns between two IPs, and cannot find more complex structural patterns. In contrast, listing the triangles of all three heavy weight edges in the data stream can find suspicious communication among three IPs and discover more suspicious structural patterns.

Use case 2: Social network data. In a social network graph, a user is considered as a node, the friend or interaction relationship between two users is considered as an edge, and the previous trust level and the number of interactions between two users can be considered as the weights of the edges. Fast community discovery in social networks is an important graph computing task, and many classical triangle-based community discovery algorithms exist, such as K-core [33] etc. Triangle counting on graph streams only enables to quickly discover the number of triangles in the graph stream for subsequent community discovery algorithms. If we can quickly enumerate the triangular structures (Triangular Vertex Connectivity) with the most active interactions in a dynamically changing social network graph [17], we can be guided to find the most active groups in the social network.

Motivated by above use cases, this paper proposes the definition of triangle weights in graph stream and proposes the problem of the top-k heavy triangle listing on graph stream. Since the graph stream in real world has the characteristics of large scale and very fast update, saving all the edges on the graph stream and performing triangle enumeration to get the accurate information of top-k heavy triangles is high in time and space complexity. Therefore, this paper proposes the approximation algorithm for top-k heavy triangle listing on graph stream: only a small set of heavy edges in graph stream is saved as the candidate edge set, and triangle enumeration is performed in the candidate edge set. This approximation algorithm can be used without storing all edges of the graph stream, and only uses a small time and space complexity to perform the top-k heavy triangle listing.

For the approximation algorithm of top-k heavy triangle listing on graph stream, we propose data structures: DolhaT and Filtered DolhaT. DolhaT and Filtered DolhaT combine the Dolha data structure [34] with the SpaceSaving (SS) [6] and Filtered SpaceSaving (FSS) [7] algorithms to efficiently enumerate the top-k heavy triangles on the graph stream. Based on Filtered DolhaT, we propose another optimized data structure, Double Filtered DolhaT (DFD), which uses the lite estimator as a double filter to reduce the error of Filtered DolhaT and improve the accuracy of the data structure without increasing the time and space complexity. We have demonstrated the superiority of the DFD data structure through extensive experiments.

2 Related work

There are two areas involved into the top-k heavy triangles listing:

2.1 Top-k heavy hitters algorithms in data streams

There exists a series of heavy hitters algorithms in the study of data streams. Given a data stream S, in order to reduce the space complexity of the algorithm, the top-k heavy hitters algorithms usually save only the top-m heavy hitters ($k < m < |S|$). For each arriving item in S, an estimator is used to estimate the previously carried weight by this item in the stream.

The SpaceSaving (SS) [6] algorithm is an approximation algorithm for finding the top-k heavy item in the data stream. SS maintains a heap $m(|m| > k)$ to store the item IDs and corresponding weights of the heavy items in the data stream, and a global estimator e to estimate the weight. Given a new data stream item i, if $i \in m$, SS increases the corresponding weight w_i; if $i \notin m$ and m is not full, then i is stored with the weight of i. If $i \notin m$ and m is full, SS assigns $w_i \leftarrow e + w_i$. Then SS removes the item j with smallest non-zero weight from m and sets estimator $e = w_i$.

The SS algorithm can consistently return the top-k heavy hitters in the data stream at a very small space and time cost. However, the SS algorithm uses a single global estimator, which can lead to high errors. Therefore Filtered SpaceSaving (FSS) [7] algorithm was invented to improve the accuracy of SS algorithm. Unlike SS which uses a single global estimator, FSS builds a hash filter table H where each cell in the table is a local estimator. When an item i on the data stream arrives, FSS first uses a hash function to map i to the x of H, and then uses the local estimator e_x here to estimate the weight. FSS can achieve higher accuracy with the same memory usage as SS.

Although a large amount of heavy hitters algorithms exist, these efforts are mainly used to detect items that occur at high frequencies in the data stream. As in work such as Wavingsketch [35] and PISketch [36], each occurrence of the streaming item is considered to carry the same weight 1. Each time an item in the data stream arrives, it is added to heavy hitter set with a certain probability. Higher frequency elements have a higher probability of being recorded in heavy hitter set. In graph stream model, steaming edges usually carry different weights. The algorithms that target the discovery of high-frequency elements are not applicable to the graph stream model. Some work focuses on the discovery of heavy hitters in sliding windows [37, 38], We do not consider the sliding window model in this paper. Therefore, we use the SS and FSS algorithms to construct the data structures proposed in this paper.

2.2 Triangle counting algorithms in graph streams

Triangles, as an important structure on graphs, have many applications in scenarios such as community in social networks [14], topic [15] and anomaly detection in web traffic networks [16]. Thus a large amount of work exists on graph stream for the triangle counting problem. However, the time complexity of triangle counting on graphs is $O(|E|^{1.5})$ [39], and triangle counting algorithms on static graphs [40] are often unable to meet the need for high-speed updates of graph streams. Therefore triangle counting algorithms on graph streams usually use sampling, where a certain number of edges are extracted from the graph stream and added to the candidate set. The number of triangles generated in the candidate set and the probability analysis of sampling are used to estimate the number of triangles that appear in the graph stream. Different works use different sampling methods and probability calculations: MASCOT [18] uses independent sampling with a fixed probability to sample the edges on the graph stream; TRIEST [12]and WRS [19] use Reservoir Sampling [20] and Random Pairing [21] to sample edges; PartitionCT [22] and subsequent works [23, 24] use Priority Sampling combined with Streaming Hyperloglog algorithm [25] to solve the Duplicated Edge problem; the study by Pavan A et al. [26] used sampling based on neighbor information; Jha M et al.'s study [27] used bilateral sampling; Ahmed N K et al.'s study [28] proposed a general sampling approach based on graph stream.

These triangle algorithms can only return an approximation of the number of triangles that appear in the graph stream and cannot enumerate the specific construction information of these triangles for further study.

3 Problem definition

Since the direction of the edges is usually not considered in the study of triangles, this paper uses the undirected graph stream model, defined as follows.

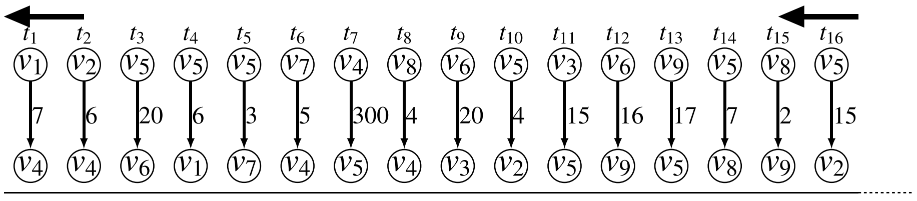
Definition 1 (Graph Stream) A graph Stream $\mathscr{G}$ is an undirected graph formed by a continuous and time-evolving sequence of edges $\{\sigma_1, \sigma_2, ...\sigma_x\}$. Each edge σ_i has node ID u, node ID v and weight w_i, denoted as $\sigma_i(\overline{uv}, w_i)$, $i = 1, ..., x$.

Definition 2 (Edge Weight) An edge $\overline{uv}$ may appear in $\mathscr{G}$ multiple times with different positive weights at different time. Each occurrence of $\overline{uv}$ is denoted as $\sigma^j(\overline{uv}, w^j)$, $j = 1, .., n$. The edge weight of $\overline{uv}$ in graph stream $\mathscr{G}$ is the weight sum of all occurrences in $\mathscr{G}$, denoted as

$$W(\overline{uv}) = \sum w^j.$$

In previous work on triangle counting [41–47], there are different ways of defining the weights of triangles. In some graph model research works [43, 46] that only have node weights and do not consider edge weights, the weight of a triangle is equal to the sum of the weights of the three nodes that make up the triangle. In some graph model research works [44, 46] that consider only edge weights, the weight of the triangle is equal to the sum of the weights of the three edges that make up the triangle. In graph stream models, we usually only consider the edge weight. Therefore, we first consider five definitions of triangle weights that are defined by the weights of the three edges. Given a graph stream (Figures 1 and 2), the top-3 heavy triangles in different definitions are shown as follows:

1. **Product of the three edge weights:** $\Delta_{\overline{v_2v_4v_5}}$, $\Delta_{\overline{v_1v_4v_5}}$, $\Delta_{\overline{v_4v_5v_8}}$.
2. **Sum of the three edge weights:** $\Delta_{\overline{v_2v_4v_5}}$, $\Delta_{\overline{v_1v_4v_5}}$, $\Delta_{\overline{v_4v_5v_8}}$.
3. **Average of the three edge weights:** $\Delta_{\overline{v_2v_4v_5}}$, $\Delta_{\overline{v_1v_4v_5}}$, $\Delta_{\overline{v_4v_5v_8}}$.
4. **Maximum of the three edge weights:** $\Delta_{\overline{v_2v_4v_5}}$, $\Delta_{\overline{v_4v_5v_8}}$, $\Delta_{\overline{v_1v_4v_5}}$.
5. **Minimum of the three edge weights:** $\Delta_{\overline{v_5v_6v_7}}$, $\Delta_{\overline{v_3v_5v_6}}$, $\Delta_{\overline{v_2v_4v_5}}$.



t_1 t_2 t_3 t_4 t_5 t_6 t_7 t_8 t_9 t_{10} t_{11} t_{12} t_{13} t_{14} t_{15} t_{16}
v_1 v_2 v_5 v_5 v_5 v_7 v_4 v_8 v_6 v_5 v_3 v_6 v_9 v_5 v_8 v_5
7 6 20 6 3 5 300 4 20 4 15 16 17 7 2 15
v_4 v_4 v_6 v_1 v_7 v_4 v_5 v_4 v_3 v_2 v_5 v_9 v_5 v_8 v_9 v_2

Figure 1 Graph stream $\mathscr{G}$

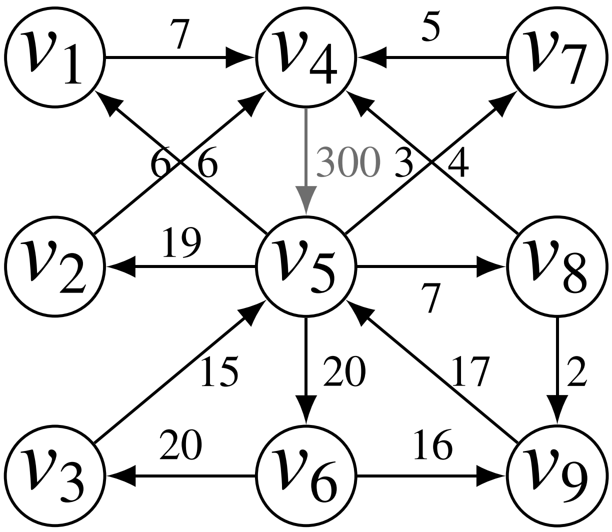


Figure 2 Graph stream topology

There is a problem in the definitions 1-4: Super Weight Edge. If one of the edges in graph stream has a very heavy weight, all triangles composed of this edge are completely dominated by this super weight edge. When we try to list the top-k heavy triangles, all the heavy triangles listed may be dominated by this Super Weight Edge, resulting in the failure to discover other triangular structures that need attention. Super Weight Edges may appear in realistic datasets. Figure 3 shows the distribution of the edge weights in the Enron email dynamic dataset [48]. The weights of the edges conform to a power-law distribution, where the weight of the edge with the highest weight reaches 2^{12}, while the weight of the third ranked edge is only 2^{10}, and the weight of the tenth ranked high-weighted edge decreases to 2^9. If the 1-4 definitions are used, the top-k heavy triangles is dominated by the edge with the highest weight.

According to the 5th definition of triangle weights, the list of the top-3 heavy triangles in Figure 2 are $\Delta_{\overline{v_5v_6v_7}}$, $\Delta_{\overline{v_3v_5v_6}}$, $\Delta_{\overline{v_2v_4v_5}}$. Under this definition, the weight of the triangle is

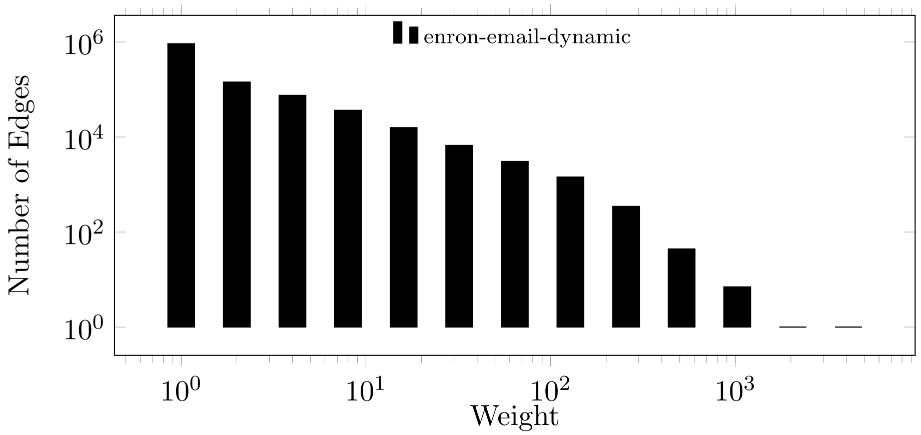


Figure 3 The edge weights distribution of enron email dynamic dataset

determined by the edge with the lowest weight, so that the query result will not be dominated by several super edges. This definition is also more in line with realistic use cases. For network traffic data, if each edge of the triangle relationship has a high weight, it is more likely that there is a suspicious pattern. For social network data, when the relationship weights among all three users are very high, it is more likely that the three users exist in a highly active community. In a real-world scenario we need to mine triangle relationships with high weights on all three edges, rather than mining triangles formed by only one very high weighted edge.

In this paper, we denote the triangle weight as the minimum weight of the three edges. The formal definition is shown as follows:

Definition 3 (Triangle Weight) *Given a set of three nodes* (u, v, w), *if the edges* $\overline{uv}$, $\overline{vw}$ *and* $\overline{wu}$ *have positive edge weights, the triangle* $\Delta_{\overline{uvw}}$ *exists in* $\mathscr{G}$. *The triangle weight of* $\Delta_{\overline{uvw}}$ *is the minimum weight of the three edges, denote as*:

$$W(\Delta_{\overline{uvw}}) = Min(W(\overline{uv}), W(\overline{vw}), W(\overline{wu}))$$

4 Approximation algorithm for top-k heavy triangles listing

Graph streams are often large in size and the time and space complexity of saving the full information for exact query is often high. Therefore, in previous algorithms for triangle counting on graph streams, the sampling of edges was used to save only a small portion of the edges in the stream, and the approximate number of triangles in the stream was derived by the probability of the distribution of the sampled edges in the stream. Although these algorithms only provide the approximate number of triangles in the graph stream, they cannot provide specific information about these triangles. However, these algorithms provide an idea: calculate the distribution of all triangles in the graph stream by a small number of triangles formed by a small set of candidate edges. This paper inherits this idea.

Lemma 1 *Given the edge set D which stores top-m heavy edges of graph stream* $\mathscr{G}$ *and the triangle set T which stores the triangles formed by D. Using the triangle minimum weight edge to define the weight of triangles, the top-k* $(k \leq |T|)$ *heavy triangles of* $\mathscr{G}$ *must exist in T.*

Proof The weight of any edge σ not in D is less than the weight of any edge in D. Since the weights of all the triangles formed by σ are less the weight of σ, the weights of all the triangles formed by σ are less than the weight of any triangle in T. Therefore, the top-$|T|$ heavy triangles are all in T. If $(k \leq |T|)$, the top-k heavy triangles of $\mathscr{G}$ must exist in T. □

Based on Lemma 1, we propose an algorithm to enumerate the top-k heavy triangles in the graph stream: filter the heavy edges of graph stream into the candidate set D and only enumerate the triangles formed by the edges in D. Since we do not need to enumerate the triangles on the whole graph stream, this algorithm can greatly reduce the space and time complexity of top-k heavy triangle listing. However, the exact extraction of the heavy edges in the graph stream requires saving all the edges in the stream and sorting them according to their weights. The space and time complexity of storing and sorting all the edges of graph stream extremely high, so we can use the heavy hitter approximation algorithms in the data stream, such as the SS and FSS algorithms, to save the approximate set of candidate edges, which cannot obtain exact results but can greatly reduce the space and time complexity of the algorithm.

In this algorithm, the capacity m of the heavy edge candidate set D determines the speed of the algorithm operation and the memory space required. If m is chosen to be small, it will cause the problem of not being able to return top-k heavy triangles, in addition to reducing the accuracy of the system. Since the top-k heavy triangles are formed by the edges in D, if the edges in D cannot form k or more than k triangles, the top-k heavy triangles cannot be returned. Although we can dynamically scale the size of m during the processing of the graph stream, it often requires additional computational cost when scaling. Therefore, the predicted size of m based on the predicted graph stream information can avoid the cost of dynamic expansion. For this problem, the following analysis is given in this section.

Lemma 2 *Given the edge set D with maximum capacity m, the edge number $|E|$ of graph stream $\mathscr{G}$ and the triangle number $|\Delta|$ of graph stream $\mathscr{G}$, when $|D| = m \geq \sqrt[3]{\frac{k}{|\Delta|}} \times |E|$, the expectation number of triangles in T is more than or equal to k.*

Proof The probability of triangles appearing in the graph stream $\mathscr{G}$, i.e., the probability that three randomly selected edges can form a triangle, is $\frac{|\Delta|}{\binom{|E|}{3}}$. Assuming that the triangles are equally distributed in the graph stream G and the candidate set D, if k triangles are expected, $\frac{k}{|\Delta|} \times \binom{|E|}{3}$ edge combinations are needed, i.e.,

$$\binom{|D|}{3} = \frac{k}{|\Delta|} \times \binom{|E|}{3}.$$

This equation can be derived to $|D| \leq \sqrt[3]{\frac{k}{|\Delta|}} \times |E|$. Therefore, when $|D| = m \geq \sqrt[3]{\frac{k}{|\Delta|}} \times |E|$, the expectation number of triangles in T is more than or equal to k.

Due to the nature of graph stream, it is impossible to know exactly the size $|\Delta|$ in the graph stream. But most graph data such as social network and communication network graphs have a large number of triangles and k is often small. In most of cases, $k \ll |\Delta|$, i.e.,$\sqrt[3]{\frac{k}{|\Delta|}} \ll 1$. According to the experiment datasets in this paper, when $k = 30$, the maximum value of $\sqrt[3]{\frac{k}{|\Delta|}}$ is about $\frac{1}{100}$ and the minimum value is about $\frac{1}{1000}$. $O(|D|)$ is usually about 2 to 3 orders of magnitude smaller than $O(|E|)$.

5 DolhaT and filtered DolhaT structure

5.1 DolhaT structure

In order to enumerate the top-k heavy triangles on graph stream, we propose the data structure called DolhaT. DolhaT combines the SpaceSaving (SS) algorithm commonly used on data stream with the Dolha graph stream data structure [34]. DolhaT uses the SS algorithm

Table 1 Notations of DolhaT

Notation	Description
$\mathcal{G}$	Graph stream
v	Node
$\sigma(\overline{uv})$	Edge between node u and v
$W'(\sigma)$	Weight of streaming edge σ
D	Candidate set of heavy edges on $\mathcal{G}$
m	Maximum capacity of D
$W(\sigma)$	Weight of edge σ stored in D
σ_{min}	Edge with smallest weight in D
T	Top-k heavy triangle set on $\mathcal{G}$
$\Delta_{\overline{uvw}}$	Triangle formed by $\overline{uv}$, $\overline{vw}$ and $\overline{uw}$
$W(\Delta_{\overline{uvw}})$	Weight of triangle $\Delta_{\overline{uvw}}$
Δ_{min}	Triangle with smallest weight in T

Table 2 Node hash table of DolhaT at t_{10}

Hash Index	0	1	2	3	4	5	6	7
Node Table Pointer	4	1	3	5	6	7	0	2

to filter out the high-weighted edges and stores the high-weighted edges in the set of candidate edges, and run the triangle enumeration algorithm on these candidate edges. DolhaT uses the combination of orthogonal list and hash table to store the candidate edges that each node and edge can be located in $O(1)$ time complexity and the neighbors of each node can be enumerated in $O(d)$ time complexity (d is the average degree of the candidate set). DolhaT also supports update replacement of candidate high-weight edges by a single linked table to sort the set of candidate edges by their weights. The data structure of DolhaT is defined as follows and Table 1 shows the notations of DolhaT.

- Optimized Dolha structure for the high-weight edges, denote as D with maximum capacity m. D has following structures:
  - Node hash tables and edge hash tables for recording the indexes of nodes and edges.
  - Node and edge tables for recording the information of nodes and edges.
    - In the node table, each cell stores the node ID v and the start point of v's neighbor list.
    - In the edge table, each cell stores the node indexes of the edge $\overline{uv}$, the edge weigt, the pointers of the neighbor list and the pointer of the weight list.
- The edge σ_{min} with minimum weight in D. The weight of σ_{min} is $W(\sigma_{min})$.
- Array for the heavy triangles, denote as T. T is sorted by the triangle weight.
- The triangle Δ_{min} with minimum weight in T. The weight of Δ_{min} is $W(\Delta_{min})$.

For example, Tables 2-6 show DolhaT data structure of Figure 1 at t_{10}. The node hash table (Table 2) and the edge hash table (Table 4) record the edge/node table indexes of candidate edges and associated nodes. Through the hash table, the given point or edge can be located in $O(1)$ time complexity. For example, the node v_1 is mapped to the cell 6 of the node hash table by the hash function $H(*)$. In the cell 6 of the node hash table, the node table pointer of v_1 is stored as 0. the edge $\overline{v_1v_4}$ is mapped to the cell 7 of the edge hash table by the hash function $H(*)$. In the cell 7 of the edge hash table, the edge table pointer of $\overline{v_1v_4}$ is stored as 0.

In the node table (Table 3), each cell stores the node ID and the pointer of the node adjacency neighbor list in the edge table. For example, the cell 0 of node table stores the node ID v_1 and the pointer of the adjacency neighbor list, which points to cell 0 in the edge table.

The edge table (Table 5) stores the indexes of the two nodes of the edge, the weight of the edge, the pointer to the weight sorted linked list, and the pointers of the two adjacency neighbor lists where this edge is located. By traversing the weight-ordered linked list from σ_{min}, all the edges in D are sorted by the weight. By traversing the adjacency neighbor list, the neighbors of each node can be enumerated.

For example, the cell 0 of the edge table stores the information of edge $\overline{v_1v_4}$: the two node indexes 0 (v_1) and 1 (v_4), the edge weight 7, the next edge pointer of the weight-ordered linked list stored as 8 and the two sets of pointers (Adjacency List L and Adjacency List R) for the two adjacency neighbor lists. The edge $\overline{v_1v_4}$ is located on the adjacency neighbor lists of v_1 (Adjacency List L) and v_4 (Adjacency List R). In Adjacency List L of cell 0, the pointers to the previous and next edges on the adjacency list of v_1 are stored as / and 3. In Adjacency List R of cell 0, the pointers to the previous and next edges on the adjacency list of v_4 are stored as / and 1. By using Adjacency List L, the related edges of v_1 can be enumerated: $\overline{v_1v_4}$, $\overline{v_1v_5}$. By using Adjacency List R, the neighbors of v_4 can be enumerated: $\overline{v_1v_4}$, $\overline{v_2v_4}$, $\overline{v_4v_7}$, $\overline{v_4v_5}$ and $\overline{v_4v_8}$.

The triangle table stores the top-k triangles on the graph stream. For example, Table 6 stores the top-3 heavy triangles $\Delta_{\overline{v_1v_4v_5}}$,$\Delta_{\overline{v_2v_4v_5}}$ and $\Delta_{\overline{v_4v_5v_7}}$ at t_{10} and the triangles are sorted by their weights.

Table 3 Node Table of DolhaT at t_{10}

Node Table Index	0	1	2	3	4	5	6	7
Node ID	v_1	v_4	v_2	v_5	v_6	v_7	v_8	v_3
Adjacency List Pointer	0	0	1	2	2	5	7	8

Table 4 Edge hash table of DolhaT at t_{10}

Hash Index	0	1	2	3	4	5	6	7	8	9
Edge Table Pointer	2	6	5	8	4	8	1	0	3	5

Table 5 Edge Table of DolhaT at t_{10}

	$\overline{v_1v_4}$		$\overline{v_2v_4}$		$\overline{v_5v_6}$		$\overline{v_1v_5}$		$\overline{v_5v_7}$		$\overline{v_4v_7}$		$\overline{v_4v_5}$		$\overline{v_4v_8}$		$\overline{v_3v_6}$		$\overline{v_2v_5}$	
Edge Index	0		1		2		3		4		5		6		7		8		9	
Node Index	0	1	1	2	3	4	0	3	3	5	1	5	1	3	1	6	4	7	2	3
Edge Weight	7		6		20		6		3		5		300		4		20		4	
Weight-Ordered List	8		0		6		1		9		3		/		5		2		7	
Adjacency List L	/	3	0	5	/	3	0	/	3	6	1	6	5	7	6	/	2	/	1	/
Adjacency List R	/	1	/	9	/	8	2	4	/	5	4	/	4	9	/	/	/	/	6	/

Table 6 Top-3 Triangles Table of DolhaT at t_{10}

	$\Delta_{\overline{v_1v_4v_5}}$			$\Delta_{\overline{v_2v_4v_5}}$			$\Delta_{\overline{v_4v_5v_7}}$		
Node Pointer	0	1	3	1	2	3	1	3	5
Triangle Weight	6			4			3		

For each new incoming edge $\sigma(\overline{uv})$ with weight $W^{'}(\sigma)$, if $\sigma \in D$ and has weight $W(\sigma)$, the weight of σ is set as $W(\sigma) = W(\sigma) + W^{'}(\sigma)$; if $\sigma \notin D$ and $|D| < m$, DolhaT adds the new edge σ into D and the weight is set as $W(\sigma) = W^{'}(\sigma)$; if $\sigma \notin D$ and $|D| = m$, DolhaT removes the edge σ_{min} and adds the new edge σ into D. Applying the similar algorithm as SS, DolhaT use the minimum weight $W(\sigma_{min})$ in D as the estimator to estimate the weight of incoming new edge σ and set the weight as $W(\sigma) = W^{'}(\sigma) + W(\sigma_{min})$. Then DolhaT resets the minimum weight edge σ_{min}.

Then DolhaT enumerates the triangles brought by the new incoming edge σ. If $W(\sigma) < (\Delta_{min})$, the update is finished. If $W(\sigma) \geq W(\Delta_{min})$, DolhaT checks the adjacency neighbor list of v. For each neighbor w of v, if $W(\overline{vw}) \geq W(\Delta_{min})$, DolhaT checks if there is an edge $\overline{vw}$. If the edge $\overline{vw}$ exists and has $W(\overline{vw}) \geq W(\Delta_{min})$, DolhaT adds the triangle $\Delta_{\overline{uvw}}$ into T. If $|T| \leq m$, the update is finished; if $|T| > m$, DolhaT removes the minimum weight triangle in T until $|T| = m$, then the update is finished. Then DolhaT resets the minimum weight triangle Δ_{min}.

Table 7 and 8 show the DolhaT at t_{11}. There are 10 candidate edges are stored in the D and the top-3 heavy triangles are stored in the T. At t_{11}, the new edge $\overline{v_3v_5}$ arrives. Since $|T| = m$ and $\overline{v_3v_5}$ is not in D, DolhaT uses the minimum weight as the estimator to estimate the weight of $\overline{v_3v_5}$ as 18 and replaces the minimum weight edge $\overline{v_5v_7}$ with $\overline{v_3v_5}$. Then we add the new triangle $\Delta_{\overline{v_3v_5v_6}}$ into T and remove the minimum weight triangle $\Delta_{\overline{v_4v_5v_7}}$.

Algorithm 1 shows the pseudocode of DolhaT. By using hash table, the time complexity to check if the incoming edge σ exists in D (line 3) is $O(1)$. If edge σ exists, DolhaT updates the weight record of σ (line 4) and updates the associated triangles in T (line 5-6). The time complexity is $O(|T|)$. If σ does not exist, DolhaT checks if $|D| = m$ (line 8). If $|D| = m$, use the estimator to estimate the weight of σ and remove the minimum weight edge σ_{min} (line 9-11). Then DolhaT adds σ into D and enumerates triangles

Table 7 Edge Table of DolhaT at t_{11}

	$\overline{v_1v_4}$		$\overline{v_2v_4}$		$\overline{v_5v_6}$		$\overline{v_1v_5}$		$\overline{v_3v_5}$		$\overline{v_4v_7}$		$\overline{v_4v_5}$		$\overline{v_4v_8}$		$\overline{v_3v_6}$		$\overline{v_2v_5}$	
Edge Index	0		1		2		3		4		5		6		7		8		9	
Node Index	0	1	1	2	3	4	0	3	3	7	1	5	1	3	1	6	4	7	2	3
Edge Weight	7		6		20		6		18		5		300		4		20		4	
Weight-Ordered List	4		0		6		1		8		3		/		5		2		7	
Adjacency List L	/	3	0	5	/	3	0	/	3	6	1	6	5	7	6	/	2	/	1	/
Adjacency List R	/	1	/	9	/	8	2	4	/	8	4	/	4	9	/	/	/	/	6	/

Table 8 Top-3 Triangles Table of DolhaT at t_{10}

	$\Delta_{\overline{v_3v_5v_6}}$			$\Delta_{\overline{v_1v_4v_5}}$			$\Delta_{\overline{v_2v_4v_5}}$		
Node Index	3	4	7	0	1	3	1	2	3
Triangle Weight	18			6			4		

```
Input: Graph Stream $\mathcal{G}$
Input: $D$: candidate edges set with maximum capacity $m$
Input: $\sigma_{min}$: the minimum weight edge in $D$
Input: $T$: top-$k$ heavy triangles
Input: $\Delta_{min}$: the minimum weight triangle in $T$
Output: The updated DolhaT
Function Main:
    for For each incoming edge $\sigma_i(\overline{uv}, W^{'}(\sigma_i))$ in $\mathcal{G}$ do
        if $\sigma_i \in D$ with $W(\sigma_i)$ then
            $W(\sigma_i) = W(\sigma_i) + W^{'}(\sigma_i)$ ;
            for each triangle $\Delta_{\sigma_i}$ associated with $\sigma_i$ in $T$ do
                Recalculate $W(\Delta_{\sigma_i})$;
        else
            if $|D| = m$ then
                $W^{'}(\sigma_i) = W^{'}(\sigma_i) + W(j)$ ;
                $D \leftarrow D \setminus \sigma_{min}$
            $D \leftarrow D \cup \sigma_i$ ;
            TriangleFind($\sigma$);
            $T \leftarrow T \cup T_{\sigma_i}$ ;
            while T $> k$ do
                $T \leftarrow T \setminus \Delta_{min}$
Function TriangleFind($\sigma$):
    $T_\sigma = \emptyset$ ;
    for each triangle $\Delta_\sigma$ associated with $\sigma$ do
        if $W(T_\sigma) > \Delta_{min}$ then
            $T_\sigma \leftarrow T_\sigma \cup \{\Delta_\sigma\}$ ;
    return $T_\sigma$;
```

Algorithm 1 DolhaT edge update.

associated with σ (line 12). In this step, the time complexity is $O(|D|)$. Then DolhaT sorted the heavy triangle array and removes the minumum weight triangle until $|T| = k$. In this step, the time complexity is $O(|T| \times \log |T|)$. If k is small and $|T| \ll |D|$, the time complexity of DolhaT update is $O(|D|)$. Otherwise, the time complexity of DolhaT update is $O(|T| \times \log |T|)$.

5.2 Filtered DolhaT structure

Filtered Spacesaving (FSS) uses a hash table approach to optimize the Spacesaving (SS) algorithm. SS algorithm uses the minimum weight in the array as the global estimator and the estimation method causes a large bias that affects the accuracy of the final result. FSS algorithm, on the other hand, uses a hash table to store a series of local estimators. The local estimators spread the values of the estimators in a more accurate way and thus improve the accuracy. This optimization strategy can also be used in DolhaT by replacing the edge hash table with a hash filter table to the candidate edges set D. Therefore, in this paper, we propose an optimized data structure Filtered DolhaT by adding the following structure. Table 9 shows the added notations of Filtered DolhaT.

Table 9 Notations of filtered DolhaT

Notation	Description
H	Hash filter table
$H(*)$	Hash function to map the streaming edge into H
e_i	Local estimator for the edge mapped into the cell i of H
f_i	Bit flag to show if any edge mapped to i is stored in D
c_i	Counter to store the number of the edges mapped into the cell i of H

- A hash filter table denoted as H. The cell i of H has a local estimator e_i, a bit flag f_i and a local counter c_i
  - The local estimator e_i stores the weight of the edges mapped to i.
  - The bit flag f_i shows if any edge mapped to i is stored in D.
  - If $f_i = 0$, the local counter c_i stores the number of the edges mapped to i. If two edges have the same estimated weight, the edge with smaller local counter is considered to be the heavier edge.
  - If $f_i = 1$, the local counter c_i is transformed into the edge table pointer and point to the edge stored in D.

Tables 10 to 12 show the structure of Filtered DolhaT at t_{13}. Filtered DolhaT has the similar structure of D except the edge hash table. To reduce redundancy, we do not describe the specific structure of D and use the schematic tables for the examples. In Table 10, the flag $f_0 = 1$ which means that c_0 is the pointer to the edge table cell 0. The flag $f_3 = 0$ which means that c_3 is the local counter of the edges mapped to 3.

For each new incoming edge $\sigma(\overline{uv})$ with weight $W'(\sigma)$, if $\sigma \in D$ and has weight $W(\sigma)$, the weight of σ is set as $W(\sigma) = W(\sigma) + W'(\sigma)$; if $\sigma \notin D$ and $|D| < m$, Filtered DolhaT adds the new edge σ into D and the weight is set as $W(\sigma) = W'(\sigma)$; if $\sigma \notin D$ and $|D| = m$, Filtered DolhaT maps edge σ_{min} into the cell x of H and sets $W(\sigma) = e_x + W'(\sigma)$. Then Filtered DolhaT compares $W(\sigma)$ with $W(\sigma_{min})$. If $W(\sigma) < W(\sigma_{min})$ or $W(\sigma) = W(\sigma_{min})$ and $c_x \geq c_{min}$, Filtered DolhaT sets $e_x = W(\sigma_{min})$ and $c_x = c_x + 1$. Otherwise, Filtered DolhaT inserts σ into D and removes σ_{min} from D. Then Filtered DolhaT maps σ_{min} to y of H and set $e_y = e_y + W(\sigma_{min})$ and $c_y = c_y + 1$. If $W(\sigma)$ is inserted into D, Filter DolhaT enumerates the triangles associated with $W(\sigma)$ and updates T in the same way as DolhaT.

Filtered DolhaT not only increases the accuracy of the results, but also reduces the computational cost of each edge update. In DolhaT, since there is only one estimator, unless the newly inserted edge has been recorded, each update triggers an edge insertion and the enumeration of triangles. In contrast, in the update process of Filtered DolhaT, only when a

Table 10 Hash filter table of filtered DolhaT at t_{13}

$H(\sigma)$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
e	0	0	0	7	0	0	0	0	0	0	0	0	4	0	0	0	0	0	0	0
f	1	0	1	0	0	1	1	0	1	0	1	0	0	1	0	1	0	1	0	1
c	8	0	4	2	0	1	3	0	7	0	5	0	1	9	0	0	0	2	0	6

Table 11 Edge table of filtered DolhaT at t_{13}

	$\overline{v_1v_4}$	$\overline{v_2v_4}$	$\overline{v_5v_6}$	$\overline{v_1v_5}$	$\overline{v_3v_5}$	$\overline{v_4v_7}$	$\overline{v_4v_5}$	$\overline{v_6v_9}$	$\overline{v_3v_6}$	$\overline{v_5v_9}$
Weight	7	6	20	6	15	5	300	16	20	17
c	0	0	0	0	0	0	0	0	0	0

Table 12 Top-3 heavy triangles of filtered DolhaT at t_{13}

	$\Delta_{\overline{v_5v_6v_9}}$	$\Delta_{\overline{v_3v_5v_6}}$	$\Delta_{\overline{v_1v_4v_5}}$
Weight	16	15	6

new heavy edge appears, the edge needs to be inserted into the candidate set and the triangle enumeration is performed Table 11.

We use the same example of Figure 1. Given a hash function $H(*)$,we have the following mappings: $3 \leftarrow H(\overline{v_5v_7})$, $3 \leftarrow H(\overline{v_4v_8})$, $12 \leftarrow H(\overline{v_2v_5})$, $2 \leftarrow H(\overline{v_3v_5})$, $8 \leftarrow H(\overline{v_6v_9})$, $13 \leftarrow H(\overline{v_5v_9})$. Since Filtered DolhaT uses the local estimators not the globel estimator, at t_{11}, the local estimator of $\overline{v_3v_5}$ is 0, the $\overline{v_3v_5}$ has the accurate weight 15 in Filtered DolhaT and replaces edge $\overline{v_5v_7}$. Then the new edges $\overline{v_6v_9}$ and $\overline{v_5v_9}$ also have the accurate weights stored by using the local estimators.

Algorithm 2 shows the processing of edge updates by Filtered DolhaT. Compared to DolhaT, Filtered DolhaT adds the process of filtering the edge weights by the local estimator (lines 3, 15 and 16). The time complexity of this operation is $O(1)$. Unlike DolhaT, Filtered DolhaT does not need to perform edge insertion and triangle enumeration every time. Triangle enumeration is only required when the edge passes the filter with a greater weight than the minimum edge weight in D. In other cases, it is not necessary to run the Triangle-Find function, and only the operation of time complexity $O(1)$ Therefore, in practice, the average complexity of Filtered DolhaT is lower than DolhaT.

6 Double filterd DolhaT structure

In Section 4, we analyze $O(|D|)$ is usually 2-3 orders of magnitude smaller than $O(|E|)$. But for Filtered DolhaT, there is an additional space overhead of hash filtering table. If there is a need to maintain the similar memory footprint as DolhaT, keeping the space complexity to $O(|D|)$, we can not guarantee $|H| \geq |E|$ to achieve a one-to-one hash mapping for the streaming edges. A large number of hash collisions can occur if $|H| < |E|$. Each local estimator may be the sum of weights of multiple edges, this can affect the accuracy of the structure. For this reason, we hope that we can further reduce the errors caused by such hash collisions and thus improve the accuracy.

According to the analysis in Section 3, the degrees of edges in realistic graph stream have a power-law distribution, and the maximum and minimum weights of edges can differ by multiple orders of magnitude. If different sizes of memory space are used to record different sizes of edge weights, the collision error can be further reduced and the accuracy rate can be improved within the limited memory space. Therefore, this paper proposes a new structure, Double Filtered DolhaT (DFD), to optimize the Filtered DolhaT structure. Double Filtered DolhaT, which is based on the Filtered DolhaT structure, changes the counter c_i to a lite estimator array b_i. Each lite estimator array b_i is divided into $|b|$ lite estimators

```
Input: Graph Stream $\mathscr{G}$
Input: $H$: the hash filter table for the edges
Input: $D$: candidate edges set with maximum capacity $m$
Input: $\sigma_{min}$: the minimum weight edge in $D$
Input: $T$: top-$k$ heavy triangles
Input: $\Delta_{min}$: the minimum weight triangle in $T$
Output: The updated Filtered DolhaT
1  Function Main:
2      for For each incoming edge $\sigma_i(\overline{uv}, W^{'}(\sigma_i))$ in $\mathscr{G}$ do
3          $i \leftarrow H(\sigma)$ ;
4          if $\sigma \in D$ with weight $W(\sigma)$ then
5              $W(\sigma) = W(\sigma) + W^{'}(\sigma)$ ;
6              for each triangle $\Delta_{\sigma_i}$ associated with $\sigma_i$ in $T$ do
7                  Recalculate $W(\Delta_{\sigma_i})$;
8          else
9              if $|D| < m$ then
10                 $D \leftarrow D \cup \{\sigma\}$ ;
11                 TriangleFind($\sigma$);
12                 $T \leftarrow T \cup T_{\sigma}$ ;
13                 while $|T| > k$ do
14                     $T \leftarrow T \setminus \Delta_{min}$
15             else if $W^{'}(\sigma) + e_i > W(\sigma_{min})$ or
                ($W^{'}(\sigma) + e_i = W(\sigma_{min})$ and $c_i < c_{min}$) then
16                 $W(\sigma) = W^{'}(\sigma) + e_i$ ;
17                 $D \leftarrow D \cup \{\sigma\}$ ;
18                 TriangleFind($\sigma$);
19                 $T \leftarrow T \cup T_{\sigma}$ ;
20                 while $|T| > k$ do
21                     $T \leftarrow T \setminus \Delta_{min}$
22                 $x \leftarrow H(\sigma_{min})$ ;
23                 $e_x = e_x + W(\sigma_{min})$ ;
24                 $c_x += 1$;
25                 $D \leftarrow D \setminus \sigma_{min}$;
26             else
27                 $e_i = e_i + W^{'}(\sigma)$;
28                 $c_i += 1$;
```

Algorithm 2 Filtered DolhaT edge update.

and the size of each lite estimator is $32/|b|$ bits (we consider the size of each counter is 32 bits). The maximum value of the lite estimator is denoted as B and the lite estimator array b_i has the same memory cost as the counter c_i. Table 13 shows the added notations of DFD.

After each edge in the graph stream is mapped to a location x of the hash estimator table H , it is also mapped to a location b_x^p of the lite estimator array b_x by another hash function. When the edge σ in the graph stream is mapped to b_x^p and $b_x^p < B$, DFD uses the b_x^p as the

Table 13 Notations of DFD

Notation	Description
b_x	Lite estimator array for the edges mapped into the cell x of H
$h(*)$	Hash function to map the streaming edge into b_x
b_x^p	Lite estimator for the edge mapped into the cell p of b_x
$\|b\|$	Size of the lite estimator array
B	Maximum value of the lite estimator

estimator to estimate the weight of σ. The $|H|$ size filter table is divided into $|H \times b|$ size and the hash collisions are reduced significantly. When b_x^p reaches the maximum value, the overflow will be recorded in the estimator e_x which will eventually be the sum of the overflow of all the lite estimators of b_x (when the first overflow occurs, it is the maximum value of the overflow of all the lite estimators of b_x. But if the overflow continues to occur, estimator e_x eventually will be the sum of the overflow of all the lite estimators). By using the lite estimator array, the smaller edge weights are stored in the lite estimators and less affected by hash collisions. For the heavies edges, the estimated weights are only affected by the overflow fraction of the lite estimators.

For each new incoming edge $\sigma(\overline{uv})$ with weight $W'(\sigma)$, if $\sigma \in D$ and has weight $W(\sigma)$, the weight of σ is set as $W(\sigma) = W(\sigma) + W'(\sigma)$; if $\sigma \notin D$ and $|D| < m$, DFD adds the new edge σ into D and the weight is set as $W(\sigma) = W'(\sigma)$; if $\sigma \notin D$ and $|D| = m$, DFD uses $H(*)$ to map σ to x of filter table H and uses $h(*)$ to map σ to the lite estimator b_x^p of the lite estimator array b_x.

If $b_x^p < B$, b_x^p is the estimator for σ, DFD sets $W(\sigma) = b_x^p + W'(\sigma)$. If $W(\sigma) < W(\sigma_{min})$, then DFD compares $W(\sigma)$ and B. If $W(\sigma) \leq B$, DFD sets $b_x^p = W(\sigma)$. If $W(\sigma) > B$, DFD sets $e_x = MAX(e_x, W(\sigma) - B)$ and $b_x^p = B$. If $W(\sigma) \geq W(\sigma_{min})$, DFD inserts σ into D and removes σ_{min} from D.

If $b_x^p = B$, e_x is the estimator for σ, DFD sets $W(\sigma) = e_x + W'(\sigma)$. If $W(\sigma) < W(\sigma_{min})$, DFD sets $e_x = W(\sigma_{min})$. If $W(\sigma) \geq W(\sigma_{min})$, DFD inserts σ into D and removes σ_{min} from D.

By using the lite estimator array b_x, the filter space is $|b|$ times larger than $|H|$ for the edges with smaller weight than B. In Filtered DohaT, the average expectation value of the estimator e_x of H is $\frac{|E|}{|H|} \times W_{avg}$ (W_{avg} is the average edge weight of $\mathscr{G}$). In DFD, the the average expectation value of the estimator is $\frac{|E|}{|H \times b|} \times W_{avg}$ if the b_x^p does not reach the maximum value B. When the b_x^p reaches B, the average expectation value of the estimator is $\frac{|E|}{|H|} \times W_{avg} - |b - 1| \times B$. In DFD, the lite estimator guarantees that the estimated weight of the streaming edge $W(\sigma)^{DFD}$ is less than or equal to the estimated weight in Filtered DolhaT $W(\sigma)^{FD}$ and larger than or equal to the real weight $W(\sigma)^R$, i.e. $W(\sigma)^R \leq W(\sigma)^{DFD} \leq W(\sigma)^{FD}$. Algorithm 3 shows the pseudocode of DFD.

We use the same example of Figure 1. Given a hash function $H(*)$,we have the following mappings: $12 \leftarrow H(\overline{v_5v_8})$, $12 \leftarrow H(\overline{v_8v_9})$. Given another hash function $h(*)$,we have the following mappings: $0 \leftarrow H(\overline{v_5v_8})$, $3 \leftarrow H(\overline{v_8v_9})$, $1 \leftarrow H(\overline{v_2v_5})$. In this example, we set $B = 2$ and $|b| = 4$.

When edge $\overline{v_2v_5}$ is added into the hash filter table H at t_{11}, $\overline{v_2v_5}$ is mapped to 12 by hash function $H(\overline{v_2v_5})$ and 1 by hash function $h(\overline{v_2v_5})$. DFD locates the lite estimator b_{12}^1 and add the weight 4 to the lite estimator b_{12}^1. Since the lite estimator b_{12}^1 reaches B, DFD sets $b_{12}^1 = 2$ and add the overflow to the estimator $e_{12} = 2$.

```
Function Main:
    for For each incoming edge $\sigma_i(\overline{uv}, W^{'}(\sigma_i))$ in $\mathcal{G}$ do
        $i \leftarrow H(\sigma)$ ;
        $x \leftarrow h(\sigma)$ ;
        if $\sigma \in D$ with weight $W(\sigma)$ then
            $W(\sigma) = W(\sigma) + W^{'}(\sigma)$ ;
            for each triangle $\Delta_{\sigma_i}$ associated with $\sigma_i$ in $T$ do
                Recalculate $W(\Delta_{\sigma_i})$;
        else
            if $|D| < m$ then
                $D \leftarrow D \cup \{\sigma\}$ ;
                TriangleFind($\sigma$);
                $T \leftarrow T \cup T_\sigma$ ;
                while $|T| > k$ do
                    $T \leftarrow T \setminus \Delta_{min}$
            else
                if $b_i^x < B$ then
                    $W(\sigma) = W^{'}(\sigma) + b_i^x$ ;
                else
                    $W(\sigma) = W^{'}(\sigma) + B + e_i$ ;
                if $W(\sigma) > W(\sigma_{min})$ then
                    $D \leftarrow D \cup \{\sigma\}$ ;
                    TriangleFind($\sigma$) ;
                    $T \leftarrow T \cup T_\sigma$ ;
                    while $|T| > k$ do
                        $T \leftarrow T \setminus \Delta_{min}$;
                    LiteEstimatorUpdate($\sigma_{min}$);
                    $D \leftarrow D \setminus \sigma_{min}$;
                else
                    LiteEstimatorUpdate($\sigma$);
Function LiteEstimatorUpdate($\sigma$):
    $y \leftarrow H(\sigma)$ ;
    $z \leftarrow h(\sigma)$ ;
    if $b_y^z = B$ then
        $e_y = W(\sigma) + e_y$ ;
    else if $W(\sigma) + b_y^z < B$ then
        $b_y^z = W(\sigma) + b_y^z$ ;
    else
        $e_y = MAX(W(\sigma) + b_y^z - B, e_y)$ ;
        $b_y^z = B$ ;
```

Algorithm 3 DFD edge update.

At t_{14}, $\overline{v_5v_8}$ is mapped to 12 by hash function $H(\overline{v_5v_8})$ and 0 by hash function $h(\overline{v_5v_8})$. The lite estimator $b_{12}^0 = 0$ which means $\overline{v_5v_8}$ is a new edge of graph stream. DFD dose not need use the estimator to estimate the previously carried weights by $\overline{v_5v_8}$. The weight of $\overline{v_5v_8}$ is set to 7. In Filtered DolhaT, The weight of $\overline{v_5v_8}$ is set to 12, since $\overline{v_5v_8}$ and $\overline{v_2v_5}$ have a hash collision.

At t_{15}, $\overline{v_8v_9}$ is mapped to 12 by hash function $H(\overline{v_8v_9})$ and 3 by hash function $h(\overline{v_8v_9})$. The lite estimator $b_{12}^3 = 0$ which means $\overline{v_5v_8}$ is a new edge of graph stream. The weight of $\overline{v_5v_8}$ is set to 3 which is smaller the minimal weight in D. DFD adds the weight to b_{12}^3 and sets $b_{12}^3 = 2$. Then DFD processes the overflow and set $e_{12} = MAX(2, 1) = 2$. In Filtered DolhaT, The weight of $\overline{v_8v_9}$ is set to 7, since $\overline{v_8v_9}$ and $\overline{v_2v_5}$ have a hash collision. Edge $\overline{v_8v_9}$ is incorrectly added to D, producing the wrong candidate edge.

At t_{16}, edge $\overline{v_2v_5}$ arrives again. DFD uses the estimator $e_{12} = 2$ and the lite estimator $b_{12}^1 = 2$ to estimate the estimate the previously carried weights by $\overline{v_2v_5}$. The correct weight 19 is updated into D. Table 14 show the cell 12 in edge hash table H of DFD at t_{16}.

By decomposing the counters in Filtered DolhaT into a table of smaller size estimators to store multiple lite estimators, the errors due to hash collisions can be reduced. With the guarantee of false positive error (the estimator must be larger than or equal to the true value), the DFD structure ensures that the estimate of all edge weights is less than or equal to the estimate of Filtered DolhaT. DFD improves the accuracy of the data structure without adding new space and time costs, achieving performance optimization of Filtered DolhaT.

7 Experimental evaluation

7.1 Experimental setup

Datasets In this paper, we use 5 real world datasets obtained from [49] in experiments. The datasets statistics are given in Table 15.

Environment All experiments are performed on a server with dual 8-core CPUs (Intel Xeon CPU E5-2640 v3 @ 2.60GHz) and 128 GB DRAM memory, running CentOS. All the data structures are implemented in C++.

7.2 Methodology

Since the weights on the real dynamic graph datasets used are all 1, the total weight of an edge is determined by the frequency of occurrence. To better test the performance of these data structures, the experiments assign a random number from 1 to 5 to each edge in the dataset as the weight of this edge. Edges that occur more frequently are also given higher total weights. This weight assignment method does not change the distribution of edge weights and also provides an experimental data environment with different size edge weights.

Table 14 Cell 12 in Edge Hash Table H of DFD at t_{16}

$H(\sigma)$	12			
Estimator e_{12}	2			
Lite Estimator Array b_{12}	0	2	0	2

Table 15 Datesets

Dataset	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert T\rvert$	$\lvert T\rvert_{max}$	$\lvert T\rvert_{avg}$
FLICKR-GROWTH [49]	2M	33.1M	6B	7M	3K
EPINIONS-USER-RATINGS [50, 51]	756K	14M	44M	2M	58
YOUTUBE-GROWTH [52]	3.2M	12.2M	64.7M	596.6K	20
CIT-HEPPH [34]	23K	3M	934M	5M	41K
ENRON-EMAIL-DYNAMIC [48]	87K	1M	822M	30M	9K

We conducted comparison experiments among DFD, DolhaT and Filtered DolhaT. The compared experiments include edge throughput, memory usage, precision rate(PR) and average relative error (ARE). The real top-k triangles set of graph stream is θ with weights $W(\theta_1), W(\theta_2)...W(\theta_k)$; the estimated top-$k$ triangles set of graph stream is ρ with weights $\rho_1), \rho_2)...\rho_k)$. PR is $\frac{\theta\cap\rho}{k}$ and ARE is $\frac{1}{k}\times\sum_{i=1}^{k}\frac{\widehat{W(\theta_i)}-W(\theta_i)}{W(\theta_i)}$ ($\widehat{W(\theta_i)}$ is the estimated weight of θ_i). We also use the exact structure Dolha as a benchmark to compare with DFD. Four different sets of parameter settings in Table 16 are used for the experiments. Using these parameter settings, experiments were conducted on the four data structures DolhaT, Filtered DolhaT, DFD and Dolha on the five data sets in Table 15, and then the average edge throughput, memory usage, precision rate(PR) and average relative error (ARE) of the five data sets were taken for comparison.

7.3 Experiments on different DFD perimeters

Section 6 discusses the composition and update operations of the DFD data structure.The DFD forms a second layer filter by dividing the counters in the hash filter table into arrays of lite estimators. Since the average expectation of each lite estimator in b is $\frac{|E|}{|H\times b|}\times W_{avg}$ and b size is fixed at 32 bits, the maximum value of each lite estimator is $B = 2^{\frac{32}{|b|}}$. For different size settings of b, there is a different impact on ARE. Since the bits of b is fixed, if $|b|$ is larger, the maximum value B is smaller which causes more overflow. if $|b|$ is smaller, the maximum value B is larger but it causes more hash collisions and raises the average expectation of errors. Therefore, $|b|$ needs to be set according to the average edge weights. When B is in the same order of magnitude as the average weight of edges, the best accurate results can be obtained.

Figure 4 shows ARE of different $|b|$ setting. The average weights of edges in the experimental datasets are 50-100 and the fixed bits of b is 32. If we set $|b| = 2$, $B = 2^{16}$ which

Table 16 Trial parameters

	Trial 1	Trial 2	Trial 3	Trial 4
Top-k	30	30	30	30
DolhaT m	$\frac{1}{40}\times\lvert E\rvert$	$\frac{1}{20}\times\lvert E\rvert$	$\frac{1}{10}\times\lvert E\rvert$	$\frac{1}{5}\times\lvert E\rvert$
Filtered DolhaT $\lvert H\rvert$	$\frac{1}{32}\times\lvert E\rvert$	$\frac{1}{16}\times\lvert E\rvert$	$\frac{1}{8}\times\lvert E\rvert$	$\frac{1}{4}\times\lvert E\rvert$
Filtered DolhaT m	$\frac{1}{80}\times\lvert E\rvert$	$\frac{1}{40}\times\lvert E\rvert$	$\frac{1}{20}\times\lvert E\rvert$	$\frac{1}{10}\times\lvert E\rvert$
DFD $\lvert H\rvert$	$\frac{1}{32}\times\lvert E\rvert$	$\frac{1}{16}\times\lvert E\rvert$	$\frac{1}{8}\times\lvert E\rvert$	$\frac{1}{4}\times\lvert E\rvert$
DFD m	$\frac{1}{80}\times\lvert E\rvert$	$\frac{1}{40}\times\lvert E\rvert$	$\frac{1}{20}\times\lvert E\rvert$	$\frac{1}{10}\times\lvert E\rvert$

far exceeds the average weight and greatly increases the hash collisions. If we set $|b| = 16$, $B = 2^2$ which is far less than the average weight, and cannot accurately record the weight values in the lite estimators. when we set $|b| = 8$, the maximum value of each lite estimator is $B = 2^4$, which is less different from the average weight. The experiments also demonstrate that ARE is smallest when $|b| = 8$. Therefore, we used $|b| = 8$ as the default experimental parameter in all subsequent comparison experiments.

7.4 Experiments on different structures

In the experiments, we use four different experimental parameter settings to compare the performance of the four data structures in terms of edge throughput, memory usage, precision rate(PR) and average relative error (ARE). Figure 5 shows the number of edges that the four data structures can process per second. Figure 6 shows the memory usage of the four data structures. The experiment sets the memory usage of Dolha, the exact data structure, to 100% as a benchmark to compare the memory usage efficiency of the other three data structures. Figure 7 shows the PR of the top-k heavy triangle enumeration, and Figure 8 shows ARE of the heavy triangle weights.

DFD versus DolhaT In the four experimental parameter settings, we set the memory footprint of each group of DFD and DolhaT to be the same. In the case of the same memory footprint, DFD has a 2× speedup relative to DolhaT, which is due to the fact that DFD has only half size $|D|$ compared to DolhaT, and thus has a lower time complexity. As the memory footprint increases, the throughput of DolhaT drops extremely fast (as shown in Figure 5). This is due to the fact that DolhaT uses a global estimator, which causes edge replacement in D and the triangle enumeration for each incoming edge. This significantly increases the update time complexity of each edge.

For DFD and Filtered DolhaT, only the heavy edges that pass the hash filter check are added to D and a triangle enumeration is performed. The most of edges are smaller than the edge weights in D and do not require a triangle enumeration. This filtering of the candidate edges significantly improves the update performance. Due to the use of filters and local estimators, DFD is much better than DolhaT data structure in terms of PR and ARE (as shown in Figures 7 and 8).

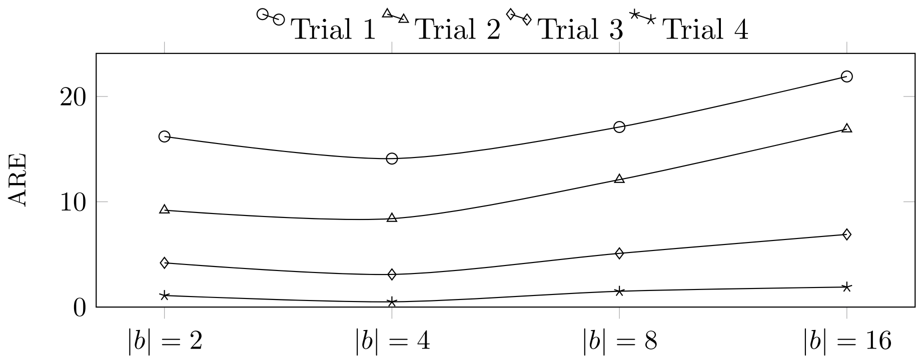


Figure 4 ARE of different $|b|$

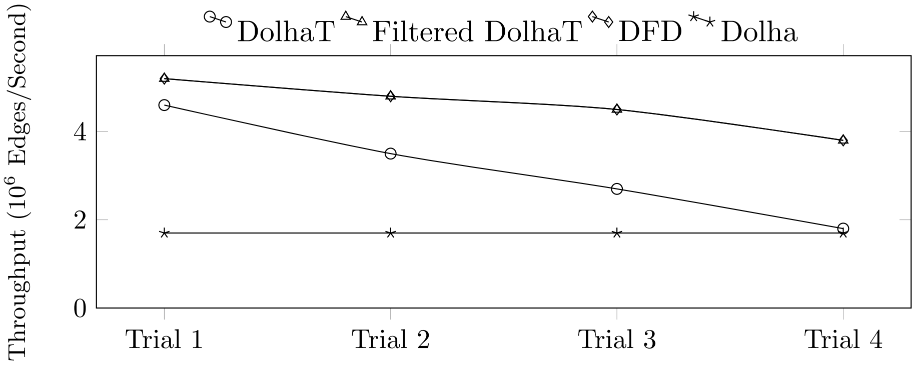


Figure 5 Throughput of different structures

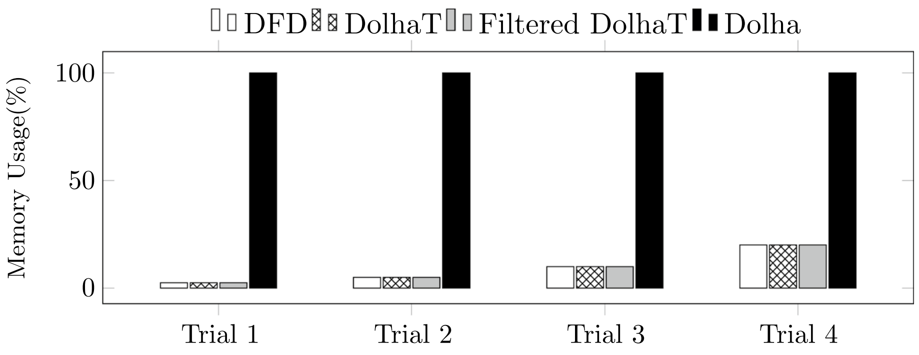


Figure 6 Memory usage of different structures

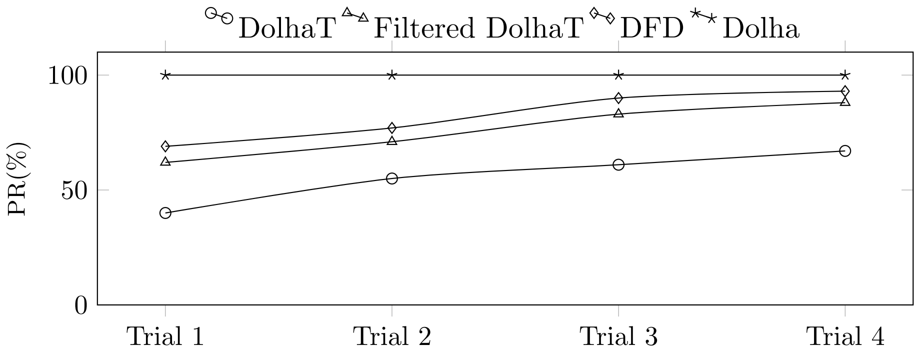


Figure 7 PR of different structures

DFD versus filtered DolhaT In the four experimental parameter settings, we set the memory allocation for each set of DFD and Filtered DolhaT to be exactly the same, and both data structures have the same size filter $|H|$ and candidate set $|D|$. In terms of throughput, both data structures are essentially the same, which is due to the fact that the update time of both data structures with exactly the same time complexity. However, PR (Figure 7) and

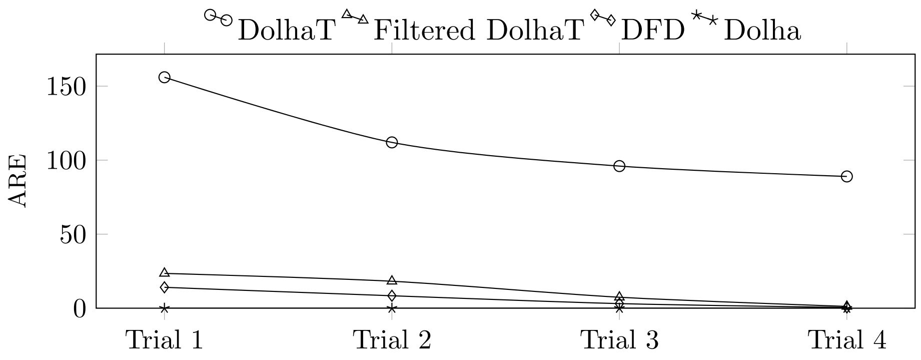


Figure 8 ARE of different structures

ARE (Figure 8) performance of DFD outperform Filtered DolhaT. This is due to the size limitation of H that leads to a large number of hash collisions. for Filtered DolhaT, an average of 4 to 32 collisions occur for each local estimator with different parameter settings, and the edge weights of these collisions are accumulated as error values in the local estimator. The edge weights of these collisions are accumulated as error values in the local estimator. In contrast, DFD uses the lite estimators to reduce the error values generated by collisions, which improves the accuracy.

DFD versus Dolha In the four experimental parameter settings, DFD occupies 1/40, 1/20, 1/10 and 1/5 of Dolha's memory and stores 1/80, 1/40, 1/20 and 1/10 of Dolha's edges. Compared to Dolha, DFD greatly reduces the memory usage overhead. Moreover, since DFD keeps fewer edges in D, DFD processes edges more efficiently, reaching a $2-3\times$ speedup. In the experiments, DFD achieves more than 90% PR with ARE of no more than 1,which means the estimated weights are no more than 2 times the true weights. The experiments show that DFD achieves high accuracy with much lower time and space cost than Dolha for top-k triangle listing.

Experiment summary The experimental results show that ARE of DFD is related to the maximum value B of the lite estimator, which is consistent with our theoretical analysis. When B is in the same order of magnitude as the average weight of edges, the best accurate results can be obtained. Since Filtered DolhaT and DFD use the local estimator based on hash tables, Filtered DolhaT and DFD possess better PR and ARE than DolhaT for the same memory usage. Since DFD uses a dual filtering structure of the local estimator and the lite estimator, DFD possesses higher accuracy and lower ARE than Filtered DolhaT. Compared to the exact structure Dolha, DFD uses smaller space and time cost and also obtains the satisfactory PR and ARE.

8 Conclusion

The triangle counting and enumeration problem is one of the fundamental problems in graph mining. As one of the simplest circle and clique structures, enumerating heavy triangles in graph stream can play an important role in use cases such as Internet attack detection, community discovery and financial fraud detection. However, traditional research on triangle computation on graph stream mainly focuses on triangle counting and lacks research on listing of heavy triangles.

In this paper, a definition of the weight of triangles is given and the rationality of this definition in practical applications is discussed. Based on this definition, this paper presents the problem of listing the top-k heavy triangles in graph stream. To solve this problem, DolhaT and Filtered DolhaT data structures are proposed in this paper by combining Dolha data structures and SpaceSaving (SS) or Filtered SpaceSaving (FSS) algorithm. These two data structures can efficiently discover the triangles with high weights in the graph flow data. In this paper, the time and space complexity of these two data structures are also analyzed. To further improve the accuracy of Filtered DolhaT, this paper proposes the data structure of DFD, which transforms the counters of Filtered DolhaT into the lite estimator array as an extra filter, and further reduces the error value of Filtered DolhaT and improves the accuracy through the double filtering method. Finally this paper verifies the superiority and effectiveness of the DFD data structure on five large-scale graph stream datasets.

In the future work we plan to further optimize the DFD data structure. On the one hand, the perimeter setting $|b|$ of DFD relies on the predicted average edge weight of graph stream. With the involvement of graph stream, the average edge weight is changing. We plan to design a self-adaptive method for DFD that $|b|$ is not fixed but automatically optimized as graph stream changes. On the other hand, by combining with other work on graph stream, the DFD data structure can be applied to more complex graph stream models, such as graph stream with sliding windows.

Acknowledgements Not applicable.

Author contributions Fan Zhang was a major contributor in the problem definition and the algorithm design and implementation as well as the manuscript writing. Xiangyang Gou revised the paper writing. Lei Zou guided the whole project and revised the paper writing. All authors read and approved the final manuscript.

Funding Not applicable.

Data availability All experimental data generated and analysed during this study are included in this published article.

Declarations

Ethical approval and Consent to participate Not applicable.

Consent for publication Not applicable.

Human and animal ethics Not applicable.

Competing interests The authors declare that they have no competing interests.

References

1. Guha, S., McGregor, A.: Graph synopses, sketches, and streams: A survey. Proc. VLDB Endow **5**(12), 2030–2031 (2012)
2. Tweet statistics. http://expandedramblings.com/index.php/march-2013-by-the-numbers-a-few-amazingtwitter-stats/10/
3. Email Statistics Report, 2015-2019. https://radicati.com/wp/wp-content/uploads/2015/02/Email-Statistics-Report-2015-2019-Executive-Summary.pdf

4. Broder, A.Z., Mitzenmacher, M.: Survey: Network applications of bloom filters: A survey. Internet Math. **1**(4), 485–509 (2003)
5. Cormode, G., Muthukrishnan, S.: An improved data stream summary: the count-min sketch and its applications. J. Algorithms **55**(1), 58–75 (2005)
6. Metwally, A., Agrawal, D., Abbadi, A.E.: Efficient computation of frequent and top-k elements in data streams **3363**, 398–412 (2005)
7. Homem, N., Carvalho, J.P.: Finding top-k elements in data streams. Inf. Sci. **180**(24), 4958–4974 (2010)
8. Afek, Y., Bremler-Barr, A., Cohen, E., Feibish, S.L., Shagam, M.: Efficient distinct heavy hitters for DNS ddos attack detection. arXiv:1612.02636 (2016)
9. Basat, R.B., Chen, X., Einziger, G., Rottenstreich, O.: Designing heavy-hitter detection algorithms for programmable switches. IEEE/ACM Trans. Netw. **28**(3), 1172–1185 (2020)
10. Newman, M.E., Watts, D.J., Strogatz, S.H.: Random graph models of social networks. Proceedings of the National Academy of Sciences **99**(suppl 1), 2566–2572 (2002)
11. Pourhabibi, T., Ong, K., Kam, B., Boo, Y.L.: Fraud detection: A systematic literature review of graph-based anomaly detection approaches. Decis. Support Syst. **133**, 113303 (2020)
12. Stefani, L.D., Epasto, A., Riondato, M., Upfal, E.: Trièst: Counting local and global triangles in fully dynamic streams with fixed memory size. ACM Trans. Knowl. Discov. Data **11**(4), 43–14350 (2017)
13. Qiu, X., Cen, W., Qian, Z., Peng, Y., Zhang, Y., Lin, X., Zhou, J.: Real-time constrained cycle detection in large dynamic graphs. Proc. VLDB Endow. **11**(12), 1876–1888 (2018)
14. Berry, J.W., Hendrickson, B., LaViolette, R.A., Phillips, C.A.: Tolerating the community detection resolution limit with edge weighting. Physical Review E **83**(5), 056119 (2011)
15. Eckmann, J.-P., Moses, E.: Curvature of co-links uncovers hidden thematic layers in the world wide web. Proceedings of the National Academy of Sciences **99**(9), 5825–5829 (2002)
16. Becchetti, L., Boldi, P., Castillo, C., Gionis, A.: Efficient algorithms for large-scale local triangle counting. ACM Trans. Knowl. Discov. Data **4**(3), 13–11328 (2010)
17. Chu, S., Cheng, J.: Triangle listing in massive networks and its applications, 672–680 (2011)
18. Lim, Y., Kang, U.: MASCOT: memory-efficient and accurate sampling for counting local triangles in graph streams, 685–694 (2015)
19. Lee, D., Shin, K., Faloutsos, C.: Temporal locality-aware sampling for accurate triangle counting in real graph streams. VLDB J. **29**(6), 1501–1525 (2020)
20. Vitter, J.S.: Random sampling with a reservoir. ACM Trans. Math. Softw. **11**(1), 37–57 (1985)
21. Gemulla, R., Lehner, W., Haas, P.J.: Maintaining bounded-size sample synopses of evolving datasets. VLDB J. **17**(2), 173–202 (2008)
22. Wang, P., Qi, Y., Sun, Y., Zhang, X., Tao, J., Guan, X.: Approximately counting triangles in large graph streams including edge duplicates with a fixed memory usage. Proc. VLDB Endow. **11**(2), 162–175 (2017)
23. Jung, M., Lim, Y., Lee, S., Kang, U.: FURL: fixed-memory and uncertainty reducing local triangle counting for multigraph streams. Data Min. Knowl. Discov. **33**(5), 1225–1253 (2019)
24. Shin, K., Oh, S., Kim, J., Hooi, B., Faloutsos, C.: Fast, accurate and provable triangle counting in fully dynamic graph streams. ACM Trans. Knowl. Discov. Data **14**(2), 12–11239 (2020)
25. Ting, D.: Streamed approximate counting of distinct elements: beating optimal batch methods, 442–451 (2014)
26. Pavan, A., Tangwongsan, K., Tirthapura, S., Wu, K.: Counting and sampling triangles from a graph stream. Proc. VLDB Endow. **6**(14), 1870–1881 (2013)
27. Jha, M., Seshadhri, C., Pinar, A.: A space efficient streaming algorithm for triangle counting using the birthday paradox, 589–597 (2013)
28. Ahmed, N.K., Duffield, N.G., Neville, J., Kompella, R.R.: Graph sample and hold: a framework for big-graph analytics, 1446–1455 (2014)
29. Yang, T., Zhang, H., Yang, D., Huang, Y., Li, X.: Finding significant items in data streams, 1394–1405 (2019)
30. Kumar, V., Sinha, D.: A robust intelligent zero-day cyber-attack detection technique. Complex & Intelligent Systems **7**(5), 2211–2234 (2021)
31. Choudhury, S., Holder, L.B., Jr., G.C., Agarwal, K., Feo, J.: A selectivity based approach to continuous pattern detection in streaming graphs, 157–168 (2015)
32. Li, Y., Zou, L., Özsu, M.T., Zhao, D.: Time constrained continuous subgraph search over streaming graphs, 1082–1093 (2019)
33. Kong, Y.-X., Shi, G.-Y., Wu, R.-J., Zhang, Y.-C.: k-core: Theories and applications. Physics Reports **832**, 1–32 (2019)

34. Zhang, F., Zou, L., Zeng, L., Gou, X.: Dolha - an efficient and exact data structure for streaming graphs. World Wide Web **23**(2), 873–903 (2020)
35. Li, J., Li, Z., Xu, Y., Jiang, S., Yang, T., Cui, B., Dai, Y., Zhang, G.: Wavingsketch: An unbiased and generic sketch for finding top-k items in data streams. In: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, pp. 1574–1584 (2020)
36. Fan, Z., Hu, Z., Wu, Y., Guo, J., Liu, W., Yang, T., Wang, H., Xu, Y., Uhlig, S., Tu, Y.: Pisketch: finding persistent and infrequent flows. In: Proceedings of the ACM SIGCOMM Workshop on Formal Foundations and Security of Programmable Network Infrastructures, pp. 8–14 (2022)
37. Song, C., Liu, X., Ge, T., Ge, Y.: Top-k frequent items and item frequency tracking over sliding windows of any size. Information Sciences **475**, 100–120 (2019)
38. Ben-Basat, R., Einziger, G., Friedman, R., Kassner, Y.: Heavy hitters in streams and sliding windows. In: IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, pp. 1–9 (2016). IEEE
39. Alon, N., Yuster, R., Zwick, U.: Finding and counting given length cycles. Algorithmica **17**(3), 209–223 (1997)
40. Schank, T., Wagner, D.: Finding, counting and listing all triangles in large graphs, an experimental study **3503**, 606–609 (2005)
41. Gall, F.L.: Improved quantum algorithm for triangle finding via combinatorial arguments, 216–225 (2014)
42. Vassilevska, V., Williams, R.: Finding a maximum weight triangle in $o(n^{3-\delta})$ time, with applications, 225–231 (2006)
43. Czumaj, A., Lingas, A.: Finding a heaviest vertex-weighted triangle is not harder than matrix multiplication. SIAM J. Comput. **39**(2), 431–444 (2009)
44. Patrascu, M.: Towards polynomial lower bounds for dynamic problems, 603–610 (2010)
45. Williams, V.V., Williams, R.: Subcubic equivalences between path, matrix and triangle problems, 645–654 (2010)
46. Williams, V.V., Williams, R.: Finding, minimizing, and counting weighted subgraphs. SIAM J. Comput. **42**(3), 831–854 (2013)
47. Williams, R.R.: Faster all-pairs shortest paths via circuit complexity. SIAM J. Comput. **47**(5), 1965–1985 (2018)
48. Cohen, W.W.: Enron email dataset. http://www.cs.cmu.edu/~enron/. Accessed in 2009
49. Rossi, R.A., Ahmed, N.K.: The network data repository with interactive graph analytics and visualization, 4292–4293 (2015)
50. Mislove, A., Koppula, H.S., Gummadi, K.P., Druschel, P., Bhattacharjee, B.: Growth of the flickr social network, 25–30 (2008)
51. Richardson, M., Agrawal, R., Domingos, P.M.: Trust management for the semantic web **2870**, 351–368 (2003)
52. Massa, P., Avesani, P.: Controversial users demand local trust metrics: An experimental study on epinions.com community, 121–126 (2005)