Effective Approximation Algorithms for Butterfly Counting in Bipartite Graph Streams

Fangyuan Zhang*, Zhongming Zuo†, Qintian Guo‡, Rui Zhang§, Sibo Wang*
*The Chinese University of Hong Kong, Hong Kong SAR, China
†Duke University, Durham, United States
‡The Hong Kong University of Science and Technology, Hong Kong SAR, China
§School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China
*{fzhang, swang}@se.cuhk.edu.hk, †zz391@duke.edu, ‡qtguo@ust.hk, §rayteam@yeah.net

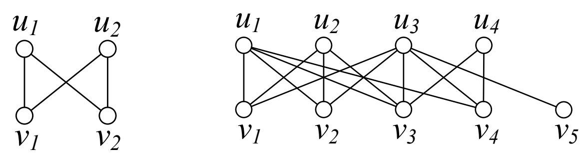
***Abstract*—Given a bipartite graph $G = \langle V = (L \cup R), E \rangle$, each edge connects a vertex in L to a vertex in R. It is widely used to characterize real-world relationships like customer-product, author-paper, etc. The butterfly motif is the smallest non-trivial motif in bipartite graphs and is crucial for the analysis of bipartite networks. Despite the significant attention that the butterfly counting problem has received in the literature, it still remains partially explored in real-world streaming scenarios where bipartite graphs are naturally modeled as graph streams over time. Existing solutions for approximate butterfly counting in bipartite graph streams may: (i) incur a high memory footprint; (ii) offer heuristic solutions without reliable performance guarantees; or (iii) mainly focus on the one-pass model, neglecting other important streaming models. These shortcomings significantly limit their broader applications.**

In this paper, we focus on three widely adopted streaming models: two-pass model, random arrival model, and sliding window model. We first develop a novel vertex sparsification technique. Specifically, by sparsifying vertices on one side, the sparsified subgraph captures more butterflies within the same expected space compared to existing methods. By subtly combining this method with the properties of the butterfly, we propose a theoretically superior algorithm for the two-pass model. Subsequently, we prove that this approach can be extended (non-trivially) to the random arrival model. Further, we innovatively incorporate priority sampling with the sparsification methods to design effective estimators for the sliding window model. Moreover, we provide a rigorous theoretical analysis for both the unbiasedness and the variance of all the proposed methods. Finally, extensive experiments are conducted on real-world bipartite graphs. The experimental results demonstrate that, with the same memory usage, the proposed algorithms achieve up to 700x lower error rates compared to state-of-the-art algorithms and produce a much higher number of valid samples.

I. Introduction

Bipartite graphs (or networks) are a pivotal structure in network modeling and are widely adopted to model relationships between two disjoint groups in many applications. For instance, in e-commerce systems, a bipartite graph can map customers to products, where each set represents customers and products, respectively. An edge is formed from a customer to a product to indicate a purchase activity [17]. Other applications of bipartite graphs can be easily found, including text networks [14], biology networks [36], and authorship networks [35]. A butterfly, also known as a (2,2)-biclique or rectangle, represents the smallest non-trivial motif in bipartite graphs (as illustrated in Figure 1 (a)). The butterfly serves as a critical metric in network analysis, such as the clustering coefficient [19], [27], which quantifies the cohesion within bipartite graphs [50]. Moreover, the count of butterflies within a graph stands as an essential metric, offering insight into the statistical relationships among vertices [24], [27]. To elucidate, the role of the butterfly in bipartite graphs is analogous to that of the triangle in general graphs; the latter being the smallest non-trivial motif in general graphs and a cornerstone in network analysis [32], [46]. However, triangles do not exist in bipartite graphs, precluding the use of triangle counting as an analytical tool for bipartite networks. Due to the importance of butterfly counting in bipartite graphs, it has attracted a plethora of works [39], [40], [48] in the literature.

In this paper we focus on the butterfly counting problem under the streaming setting, where the bipartite graph evolves over time. Such streaming scenarios are common in real-world applications. For instance, in e-commerce, transactions between consumers and products naturally generate a bipartite graph stream, with edges timestamped to reflect transaction times [52]. Similarly, in computer networks, the infrastructure involving firewalls and Intrusion Detection Systems (IDS) can be modeled as a bipartite flow graph, representing the interaction between internal and external hosts [16]. Regarding the problem of approximate triangle counting in streaming scenarios, a considerable amount of existing work has conducted in-depth studies for different common models, such as the two-pass model [5], [6], [23], random arrival model [8], [22], and sliding window model [11], [34]. In contrast, approximate butterfly counting is still under-explored under streaming scenarios. Applying existing methods to data stream problems encounters the following issues. To explain, exact counting algorithms [28], [40] on bipartite graphs require $O(m)$ space to store the entire graph for counting, where m is the size of the edge set. Consequently, this leads to a high memory cost, especially for small-memory devices such as switches, routers, and IoT devices in computer networks [16], in cases where the size of certain bipartite graph streams is unknown or even infinite. In addition, approximation algorithms [28], [51] for local sampling need to preserve all information on the graph in advance for multiple sampling iterations, which runs into similar problems. Existing approximation algorithms for



(a) Example butterfly (b) Example of a bipartite graph

Fig. 1. Bipartite graph examples

streaming scenarios, a class of empirical methods [33], cannot provide reliable theoretical guarantees, which leads to biased estimation results. Another class of sampling-based methods [16], [29] provides unbiased theoretical guarantees but lacks in-depth investigation into several classic streaming scenarios, such as two-pass model [9], [13], [21], random arrival model [7], [12], [31], [37], and the sliding window model [11], [15], [45]. While the work [29] does explore the sliding window model on graph streaming, it merely extends the hierarchical edge sampling for general graphs for estimation, resulting in unsatisfactory valid sampling rates and high estimation errors. Actually, on the aforementioned streaming settings, we may utilize the nature of bipartite graph to increase the sampling probability of a butterfly with the same expected space.

Motivated the above issues, we propose effective algorithms for approximate butterfly counting in streaming scenarios, deeply utilizing the nature of bipartite graphs and butterflies. We first propose a novel sparsification technique: by sampling vertices on one side, we generate a subgraph G' that retains all edges associated with the sampled vertices. The probability of retaining a butterfly in G' is $O(p^2)$ when sampling mp edges in expectation, where p is the sampling probability. This technique improves the probability of retaining a butterfly compared to other sparsification methods. Notably, although CAS method [16] could sample a butterfly with a probability of $O(p^2)$ in one-pass streaming, it is unable to produce a subgraph that retains all the sampled butterflies. Moreover, as an edge-based sampling method, it cannot further increase the probability of sampling a butterfly to $O(p)$. Specifically, the two sampled wedges (definition of wedge is referred to Section 2) of one butterfly are independent events, and the probability of sampling a wedge cannot be lower than that of sampling an edge; hence, this former method cannot boost the sampling probability of a butterfly to $O(p)$. As for our vertex-based sparsification technique, the technique can be applied to different streaming models with $O(p)$ probability to sample a butterfly. Since the variance of the estimate is inversely proportional to the probability of sampling a butterfly, increasing the probability that a butterfly is sampled could reduce the error [16], [28]. For the two-pass model, our algorithm VS-2PASS is able to sample a butterfly when only one of its vertices is sampled. This increases the probability of each butterfly being sampled to $O(p)$. We then carefully extend our algorithm to the random-arrival model. Under the random-arrival model, the probability of each butterfly being sampled differs from the previously proposed algorithms. After

TABLE I
FREQUENTLY USED NOTATIONS.

Notation	Description
$G = \langle V, E\rangle$	a bipartite graph that consists of vertex set V and edge set E
n, m	the number of nodes and edges, respectively
L, R	two disjoint vertex subsets of V
$e = (u, v)$	an edge e between vertices u and v
$\bowtie_G$	the number of butterflies in G
$c_{u1,u2}$	the number of wedges between u_1 and u_2
W^k	the set of the first k edges in a graph stream
$W_{t_1}^{t_2}$	the set of edges within the time range $(t_1, t_2]$
G_W	the graph formed by all the edges in set W
$hash$	a hash function can map V uniformly and independently to $(0, 1)$

analyzing the probabilities of various edge arrival types, we obtained an unbiased estimator suitable for the random-arrival model. For the sliding window model, our algorithm B-TSW utilizes the substream and priority sampling techniques. We enhance algorithm performance using vertex sparsification techniques, leading to an improved methods, VS-TSW (based on vertex sparsification). To summarize, the main contributions of this paper include:

- A novel approximate butterfly counting algorithm under two-pass model, sampling each butterfly with $O(p)$ probability.
- Non-trivial extension of the approximate butterfly counting algorithm to random arrival model.
- A novel algorithm under sliding-window model utilizing priority sampling and vertex sparsification technique.
- Rigorous theoretical analysis of the unbiasedness and variance of the proposed algorithms.
- Experiments involving large real datasets, whose results confirm that the proposed methods achieve significant performance gains in terms of accuracy and valid samples compared to existing solutions, under the same space usage.

II. Preliminaries

Consider an unweighted and undirected bipartite graph $G = \langle V = (L \cup R), E\rangle$ where V and E represent the vertex set and edge set, respectively. It has the following properties: (i) The vertex set V is divided into two disjoint sets L and R, i.e., $L \cap R = \emptyset$; (ii) The edge set $E \subseteq L \times R$, where the notation $\times$ represents the Cartesian product, namely, each edge in E connects two vertices from different sets. The set L (resp. R) is called the left (resp. right) side of the graph. Let m and n be the sizes of the edge set and the vertex set, respectively. Let (u, v) denote the edge between vertices u and v. Let $N_v = \{u|(v, u) \in E\}$ denote the set of neighbors of vertex v, and $d_v = |N[v]|$ denote the degree of v.

A. Problem Definition

Definition 1 (Butterfly). *Given a bipartite graph $G = \langle V = (L \cup R), E\rangle$ and vertices $a, b \in L$ and $c, d \in R$, if edges (a, c), (b, c), (a, d), and (b, d) are all present in E, the subgraph induced by $\{a, b, c, d\}$ forms a butterfly.*

We use $\boxtimes_G$ (or $\boxtimes$ with the subscript G omitted if the context is clear) to denote the number of butterflies in G, and Y_G to denote the estimated number of butterflies in G.

Definition 2 (Wedge). *Given a bipartite graph $G = \langle V, E\rangle$ and vertices $u_1, u_2, v \in V$, if $(u_1, v) \in E$ and $(u_2, v) \in E$, the subgraph induced by $\{u_1, u_2, v\}$ is a wedge.*

A butterfly can be regarded as the union of two interconnected wedges, namely $\{u_1, u_2, v_1\}$ and $\{u_1, u_2, v_2\}$, with u_1, u_2 on one side of G, and v_1, v_2 on the other. Let c_{u_1,u_2} denote the number of wedges between vertices u_1 and u_2. Let c_L and c_R denote the total number of wedges between all pairs of vertices in L and R, respectively, i.e., $c_L = \sum_{i\neq j \wedge u_i,u_j \in L} c_{u_i,u_j}$, $c_R = \sum_{i\neq j \wedge u_i,u_j \in R} c_{u_i,u_j}$.

Example 1. *Consider the example shown in Fig. 1 (b). Vertices u_3, v_3, and u_4 form a wedge, and vertices u_3, u_4, v_3, and v_4 form a butterfly. For the graph G, we have $\boxtimes = 14$, $c_L = 15$ and $c_R = 20$.*

A graph stream is a sequence of edges $e_1, e_2, \cdots$, where e_i is the i-th edge in the stream. Edges of the stream arrive in a sequential order. We use $t[e_i]$ to denote the arriving time of e_i ($\forall i \geq 1$, $t[e_i] > 0$ and $t[e_i] \leq t[e_{i+1}]$). For a given graph stream S, let W^k denote the edge set of the first k edges in S, i.e., $W^k = \{e_i \in S \mid 1 \leq i \leq k\}$. Let $W_{t_1}^{t_2}(t_1 < t_2)$ denote the edge set of all the edges that arrive after t_1 yet no later than t_2, i.e., $W_{t_1}^{t_2} = \{e_i \in S | t_1 < t[e_i] \leq t_2\}$. We use G_W and $\boxtimes_W$ to denote the graph formed by edge set W and the number of butterflies in G_W, respectively.

Definition 3 (Two-pass model). *In the two-pass model, an algorithm can access the graph stream twice.*

Definition 4 (Random-arrival Model). *In the random-arrival model, the edges of the graph stream arrive in a sequence that is uniformly randomly chosen from all possible orders.*

Note that, in the two-pass model, we have no assumption about the order of the edges of the graph stream, i.e., they can be in arbitrary order. In this paper, we focus on two types of window settings: the *infinite window* and the *time-based sliding window*.

Definition 5 (Infinite Window). *Given a graph stream, the infinite window contains the set of all the edges seen so far.*

Definition 6 (Time-based Sliding Window). *Given a graph stream and a window length N, at any time t, the sliding window contains the set of edges that arrived within the time range $(t - N, t]$.*

B. Related Data Structure

In this subsection, we briefly introduce two probablistic data structures that would be used in our algorithms.

AMS sketch. The AMS sketch is designed to estimate the second frequency moment (F_2), which represents the sum of squared frequencies of elements in a data stream [4]. Specifically, if the stream consists of elements from a universe $\{1, 2, \ldots, n\}$, and f_i denotes the frequency of the i-th element, then F_2 is defined as:

$$F_2 = \sum_{i=1}^{n} f_i^2.$$

The AMS sketch approximates F_2 efficiently without requiring the full frequency distribution, making it suitable for large-scale data streams. The algorithm operates as follows:

1) Initialize a set of counters, each associated with a random hash function that maps stream elements to $\{-1, 1\}$.
2) For each element in the stream, update the counters based on the hash function's output for that element.
3) Estimate F_2 by computing the average of the squared counter values.

Mathematically, the estimation of F_2 can be expressed as:

$$\hat{F}_2 = \frac{1}{m} \sum_{j=1}^{m} X_j^2,$$

where m is the number of counters, and X_j represents the value of the j-th counter after processing the stream.

The AMS sketch guarantees that $\hat{F}_2$ provides an unbiased estimate of F_2, with a variance that decreases as the number of counters, m, increases. This trade-off between accuracy and memory usage makes it an efficient solution for streaming environments where exact computation of F_2 is infeasible.

HyperLogLog Sketch. The HyperLogLog algorithm [10] is a probabilistic data structure designed to estimate the cardinality of a dataset, which is the number of distinct elements in the dataset. It is effective in big data analytics, where computing the exact count is impractical due to the dataset's large size. HyperLogLog provides an approximation with a standard error of $1.04/\sqrt{m}$, where m is the number of registers used by the algorithm. The HyperLogLog sketch make usage of hash functions to process each element in the dataset, mapping them to bit strings. The algorithm then uses the position of the leftmost '1' bit in each hashed output to estimate the cardinality. The key steps are as follows:

1) Apply a hash function to each element, producing a fixed-length bit string.
2) Divide the bit strings into m partitions based on the initial bits, corresponding to m registers.
3) For each partition, record the maximum number of leading zeros observed plus one. Store this value in the corresponding register.
4) Use the harmonic mean of the values in the registers to estimate the dataset's cardinality.

The cardinality is estimated using the formula:

$$\hat{n} = \alpha_m m^2 \left(\sum_{j=1}^{m} 2^{-M[j]} \right)^{-1},$$

where $\hat{n}$ is the estimated cardinality, m is the number of registers, $M[j]$ represents the value of the j-th register, and α_m is a correction constant that depends on m.

In this paper, we aim to design algorithms to estimate butterfly counting under streaming scenarios. In Section III, we explore how to design effective algorithms for the infinite window case with two-pass model and random-arrival model. In Section IV, we design algorithms for the time-based sliding window case.

III. Solutions for Infinite Windows

In this section, we first present our novel vertex sparsification technique that can retain each butterfly with a probability of p^2 using $O(mp)$ expected space. Then, we discuss the approximate butterfly counting problem on different models of the streaming setting. For the two-pass model, we provide a two-pass algorithm (**VS-2PASS**) based on vertex sparsification and wedge estimation. For the random-arrival model, we designed a one-pass algorithm (**VS-1PASS**) based on vertex-sparsification as well.

A. One-Sided Vertex Sparsification

The workflow of a sparsification technique can be summarized as follows: it first generates a subgraph instance G' from the input graph G, then exactly compute the number of butterflies present in G', and finally estimate the butterfly count in the original graph G according to the outcome of subgraph G'. The size of the subgraph G' is typically much smaller than that of the input graph G, enabling the efficient computation of the exact number of butterflies in G'. The key challenge for a sparsification technique lies in designing a method to generate G' as an effective representative of G, ensuring that the butterfly count in G can be accurately estimated.

One-sided Vertex Sparsification. The main idea of our sparsification technique is to sample vertices from one of the two vertex sets. Without loss of generality, we perform vertex sparsification on L, In particular, each vertex is independently sampled with probability p by $hash$[1], where p denote the successfully sampling probability. Each sampled vertex, along with all its associated edges as well as the corresponding vertices in R, are retained in the output graph G', while all other vertices and edges are discarded.

The following lemma establishes the space complexities of this algorithm.

Theorem 1. *Given graph G and probability p, Alg. 1 generated a subgraph G' with expected memory size of $O(mp)$.*

Proof. The expectation for the total number of edges to be traversed can be proved as follows. We use X_i to indicate whether i-th edge remains. Since G is a bipartite graph, an edge (u, v) must have and only one vertex that belongs to L. Clearly, the probability of the corresponding vertex being preserved is p, so the probability of this edge being preserved is also p. So $\mathbb{E}[X_i] = p$. Denote the random variable X as the number of remaining edges. By the linear of expectations, we know that

$$\mathbb{E}[X = \sum_{i=1}^{m} X_i] = \sum_{i=1}^{m} \mathbb{E}[X_i] = mp$$

This is also the expected size of the number of edges that is retained in G'. □

[1]A hash function can map V uniformly and independently to $(0, 1)$

Algorithm 1: One-sided vertex sparsification (G, p)

Input: Input graph G, probability parameter p
Output: The graph G' after vertex sparsification

```
G' ← ∅;
for vertex v ∈ L do
    if hash(v) ≤ p then
        G' ← G' ∪ {(v, u) | u ∈ N_v};
    end
end
return G';
```

Unbiased Estimator. We invoke Algorithm 1 with probability parameter p to obtain a sparsified subgraph G'. Then, we can utilize existing exact butterfly counting algorithms to compute the number of butterflies in G', e.g., `ExactBFC` [28]. With the information about the butterfly number in subgraph G', we are allowed to estimate the total number of butterflies in the original graph G.

Let ⧖ denote the number of butterflies in the original graph G. Assume that these ⧖ butterflies are numbered from 1 to ⧖. Let X_i be a random variable, such that X_i is 1 if the i-th butterfly is contained in the sparsified graph G', and 0 otherwise. Recall that each vertex in the set L of G' is sampled independently, we have

$$\Pr[X_i = 1] = p^2.$$

This is because only if both of two vertices in L of a butterfly appear in G', the butterfly is retained in G'. Let $X = \sum_{i=1}^{\text{⧖}} X_i$. Clearly, we have

$$\mathbb{E}[X] = \sum_{i=1}^{\text{⧖}} \mathbb{E}[X_i] = \text{⧖} \cdot p^2.$$

The value X can be obtained by exaclty computing the number of butterflies in the sparsified graph G', i.e., $X = \texttt{ExactBFC}(G')$. Therefore, with p^{-2} as the scaling factor, we can report $Y_{VS} = p^{-2} \cdot \texttt{ExactBFC}(G')$ as an estimate for ⧖. We then derive the unbiasedness of the estimator Y_{VS}:

Lemma 1. $\mathbb{E}[Y_{VS}] = \text{⧖}$.

Note that our vertex sparsifier retains each butterfly with probability p^2, whereas the probability that a butterfly is retained in existing methods edge sparsifier and colorful sparsifier is p^4 and p^3 [28], respectively. It implies, our sparsifier leads to more retained butterflies, with the same number of sampled edges. Next, we analyze the variance of Y_{VS}. Let

$\gamma_{1,L}$ (resp. $\gamma_{2,L}$) be the number of unordered butterfly pairs sharing one vertex (resp. two vertices) in L. We have:

Lemma 2. $\mathbb{V}\mathrm{ar}[Y_{VS}] \leq p^{-2}⧖ + p^{-1}\gamma_{1,L} + p^{-2}\gamma_{2,L}$.

Proof. Recall that X_i denotes the random variable indicating whether the i-th butterfly is retained in G' or not. By definition,

$$\begin{aligned}
\mathbb{V}\mathrm{ar}[Y_{VS}] &= \mathbb{V}\mathrm{ar}[p^{-2}\sum_{i=1}^{⧖} X_i] \\
&= p^{-4}[\sum_{i=1}^{⧖}\left(\mathbb{E}[X_i] - \mathbb{E}^2[X_i]\right) + \sum_{i\neq j}\mathbb{C}\mathrm{ov}\left(X_i, X_j\right)] \\
&= p^{-4}[⧖\left(p^2 - p^4\right) + \sum_{i\neq j}\left(\mathbb{E}[X_iX_j] - \mathbb{E}[X_i]\mathbb{E}[X_j]\right)].
\end{aligned}$$

Let us analyze the variance by considering the butterfly pair (i, j): If butterfly i and j do not share any vertex, it is impossible for $⧖_{u,v}$ to compute butterfly i and j at the same time. In this case, $\mathbb{E}[X_iX_j] = \mathbb{E}[X_i]\mathbb{E}[X_j]$. Next, we consider that butterfly i and j both contain u, v, in which case the vertices on the other side maybe three or four. In both cases, their covariances are the same and can be computed as follows: $\mathbb{E}[X_iX_j] = \Pr[X_i = 1]\Pr[X_j = 1|X_i = 1] = p^2$, $\mathbb{C}\mathrm{ov}\left(X_i, X_j\right) = p^2 - p^4$. Let $\gamma_{2,L}$ denote the number of butterfly pairs that share the same two vertices in side L. Similarly, we can consider the case that butterfly i and j share only one vertex. Let $\gamma_{1,L}$ be the number of butterfly pairs sharing one vertex in L. Then, $\mathbb{V}\mathrm{ar}[Y_{VS}]$ can be written as:

$$\begin{aligned}
&\mathbb{V}\mathrm{ar}[Y_{VS}] \\
&= p^{-4}[⧖\left(p^2 - p^4\right) + \gamma_{1,L}\left(p^3 - p^4\right) + \gamma_{2,L}\left(p^2 - p^4\right)] \\
&\leq p^{-2}⧖ + p^{-1}\gamma_{1,L} + p^{-2}\gamma_{2,L}
\end{aligned}$$

which finishes the proof. □

Remarks. Since the variance of the estimate is inversely proportional to the butterfly sampling probability [16], [28], vertex sparsification can significantly reduce the estimation error compared to colorful sparsification and edge sparsification. Compared to edge sparsification, colorful sparsification [25] increases the sampling probability by a factor of p from p^4 to p^3, with the same expected space cost. Our proposed method further increases the probability by a factor of p.

B. Two-Pass Model

In the two-pass model, we have no assumption about the order of the edges of the graph stream, i.e., it can be in arbitrary order. Our algorithm for the two-pass model is called *VS-2PASS*. The basic idea of the VS-2PASS algorithm is to apply the vertex sparsification and wedge estimation techniques to the streaming setting and utilize the properties of the two-pass model. VS-2PASS algorithm gives an estimation of the number of butterflies in the whole graph after two passes of the graph stream.

As mentioned in Section II, a butterfly consists of two wedges between the same two vertices on the same side of G. The number of butterflies in G equals the number of all such

Algorithm 2: VS-2PASS (S, p)

```
1  N_u ← ∅ for u ∈ V;
2  c_L ← 0;
3  Initialize AMS sketch;
4  for each edge(u, v) in stream S do
5      coin ← hash(u);
6      if coin ≤ p then
7          N_v ← N_v ∪ {u};
8          S ← S − {edge(u, v)};
9      end
10 end
11 for each edge(u, v) in stream S do
12     for each w ∈ N_v do
13         c_L ← c_L + 1;
14         c_{u,w} ← c_{u,w} + 1;
15         Update AMS sketch;
16     end
17 end
18 F_L ← Query value returned by AMS sketch;
19 return (F_L − c_L)/(4p − 4p^2);
```

pairs of wedges on one side of G. Without loss of generality, we continue focusing on wedges between vertices on the L side of G. For a vertex pair (u_1, u_2) in L, the number of butterflies consisting u_1 and u_2 (denoted as b_{u_1,u_2}) can be computed as

$$b_{u_1,u_2} = \binom{c_{u_1,u_2}}{2} = \frac{c^2_{u_1,u_2} - c_{u_1,u_2}}{2}.$$

where c_{u_1,u_2} denote the number of wedges between vertices u_1 and u_2. Let Θ_L denote the set of unordered vertex pairs in L. The number of butterflies in G can be computed as

$$⧖ = \sum_{(u_i,u_j)\in\Theta_L} b_{u_i,u_j} = \sum_{(u_i,u_j)\in\Theta_L} \frac{c^2_{u_i,u_j} - c_{u_i,u_j}}{2}.$$

Let $F_L = \sum_{(u_i,u_j)\in\Theta_L} c^2_{u_i,u_j}$, and $c_L = \sum_{(u_i,u_j)\in\Theta_L} c_{u_i,u_j}$, the number of butterflies in G can be computed as

$$⧖ = \frac{F_L - c_L}{2}$$

Algorithm 2 shows the pseudo-code of our VS-2PASS algorithm. Given a graph stream S, VS-2PASS processes S twice to compute the estimation of the number of butterflies in S. In the first pass, it performs vertex sampling. For each edge $e = (u, v)$ in the stream ($u \in L, v \in R$), we apply a hash function to u to decide whether to sample e (Line 3). If the hash value of u is less than or equal to the sampling probability p, e is sampled, which means the connection between v and u is recorded (Lines 4-5). If e is sampled, we delete edge e from S so that e will not be received or will be received as invalid in the second pass (Line 6).

In the second pass, VS-2PASS performs wedge estimation. After the first pass, each edge of S (except those deleted) is transmitted to the algorithm again. For each edge $e = (u, v)$, we first check each recorded neighbor of v. Note that u is not recorded as a neighbor of v because e would have been deleted otherwise. For each recorded neighbor w of v, we update the total number of wedges c_L (Line 11). In VS-2PASS, we utilize the AMS sketch to estimate the second frequency moment F_L. We have provided the details of the AMS sketch in Section II.

After counting the number of c_L and F_L, we can derive the estimation of $⧖_S$ as:

$$Y_2 = \frac{F_L - c_L}{4p - 4p^2}. \tag{1}$$

Next, we show the probability that a butterfly is sampled in the following lemma.

Lemma 3. *Algorithm 2 samples each butterfly with probability*

$$P_{⧖2} = 2p - 2p^2.$$

All the missing proofs are deferred to Section V. We can further prove that the estimator in Equation (1) is a unbiased estimator for the value $⧖_S$, and derive its variance.

Lemma 4. *The estimation of VS-2PASS with Equation 1 is unbiased:*

$$\mathbb{E}[Y_2] = ⧖_S.$$

Lemma 5. *The variance of the estimation of VS-2PASS is:*

$$\mathbb{V}\mathrm{ar}[Y_2] = ⧖_S\left(\frac{1}{2p - 2p^2} - 1\right) + \frac{\gamma_{1,L} + 2\gamma_{2,L}}{2p - 2p^2} - (2\gamma_{1,L} + 2\gamma_{2,L}),$$

where $\gamma_{1,L}$ (resp. $\gamma_{2,L}$) is the number of unordered butterfly pairs sharing one vertex (resp. two vertices),.

Space complexity. The space cost of VS-2PASS consists of two parts: (i) the space for storing sampled edges and (ii) the space for storing wedge numbers between vertices on the left side. It is clear that the space complexity for the first part is $O(pm)$ (m denotes the number of edges in S). The space for the second part is M_A, where M_A is the size of the AMS sketch. Therefore, the space complexity of VS-2PASS is $O(pm + M_A)$ when AMS sketch is used. Therefore, the space cost of VS-2PASS is the same as CAS method [16] with the same parameter.

Time complexity. For VS-2PASS, the time complexity is dominated by the time for updating wedges, which happens every time a wedge is sampled. The total number of wedges between vertices on the L side is:

$$c_L = \sum_{v_i \in R} \binom{d_{v_i}}{2} \in O\Big(\sum_{v_i \in R} d_{v_i}^2\Big).$$

Let P_Λ denote the probability of sampling a wedge. Consider the wedge $\Lambda = \{u_1, v_1, u_2\}$ in Fig 1 (a). VS-2PASS samples Λ if and only if one of u_1 and u_2 is sampled. Hence, for VS-2PASS, $P_{\Lambda 2} = 1 - p^2 - (1-p)^2 = 2p - 2p^2 \in O(p)$. Therefore, the expectation of the number of sampled wedges is in $O(p \sum_{v_i \in R} d_{v_i}^2)$. The time complexity of updating a wedge is $O(l)$ with the AMS sketch, where l is a parameter for the number of registers in the AMS sketch. Hence, the time complexity is $O(pl \sum_{v_i \in R} d_{v_i}^2)$ with AMS sketch, which is also the same as CAS method [16].

Algorithm 3: VS-1PASS (S, p)

Input: Edge stream S, vertex sampling probability p, hash function $hash : V \to (0, 1)$

Output: Estimation of $⧖$

1 $N_u \leftarrow \emptyset$ for $\forall u \in V$, $c_L \leftarrow 0$, $Initialize\ AMS\ sketch$;
2 **for** $each\ edge(u, v)\ in\ stream\ S$ **do**
3   **for** $each\ w \in N_v$ **do**
4         $c_L \leftarrow c_L + 1$;
5         $c_{u,w} \leftarrow c_{u,w} + 1$;
6   update AMS sketch;
7   **end**
8     $coin \leftarrow hash(u)$;
9   **if** $coin \leq p$ **then** $N_v \leftarrow N_v \cup \{u\}$;
10 **end**
11 $F_L \leftarrow$ Query value returned by AMS sketch;
12 **return** $(F_L - c_L)/(p + p^2)$;

C. Random-Arrival Model

Next, we consider the random-arrival model, where the edges of G arrive one by one in a sequence that is uniformly randomly chosen from all possible orders. We call our algorithm for the random-arrival model as *VS-1PASS*.

The algorithm VS-1PASS algorithm follows a similar idea of VS-2PASS to apply the vertex sparsification and wedge estimation techniques to the streaming setting. Different from previous 1-PASS solutions, we explore the randomness of the edge sequences under the random-arrival model, and analyze the probability of each butterfly being sampled leading to a more effective unbiased estimator. VS-1PASS can provide an estimation at any time while receiving edges.

Algorithm 3 shows the pseudo-code of VS-1PASS. Given a graph stream S, VS-1PASS processes S only once to compute estimations with the following steps. For each edge $e = (u, v)$ in the stream ($u \in L, v \in R$), we first check each currently recorded neighbor of v, denoted as N_v (Line 3). Clearly, u is currently not in the recorded neighbors of v since the edge connecting u and v is received for the first time and there are no multi-edges between u and v in S. For each recorded neighbor w of v, c_L, $c_{u,w}$ and F_L are updated (Lines 4-6). Then, same as in Algorithm 2, e is sampled according to the hash value of u.

We derive the estimation of $⧖_S$ as:

$$Y_1 = \frac{F_L - c_L}{p + p^2}. \tag{2}$$

Since VS-1PASS always maintains F_L and c_L for the edges it currently receives, it can give estimations for the received edges at any time during the edge transmission process.

Lemma 6. *Algorithm 3 samples each butterfly with probability*

$$P_{⧖1} = \frac{p + p^2}{2}.$$

Further, we show the estimator of VS-1PASS is unbiased and analyze its variances.

Lemma 7. *The estimation of VS-1PASS with Equation 2 is unbiased:*

$$\mathbb{E}[Y_1] = \Join_S.$$

Lemma 8. *The variance of VS-1PASS is:*

$$\mathbb{V}\mathrm{ar}[Y_1] \le \Join_S\left(\frac{2}{p+p^2}-1\right)+\frac{4\gamma_{1,L}+4\gamma_{2,L}}{p+p^2}-(2\gamma_{1,L}+2\gamma_{2,L}),$$

where $\gamma_{1,L}$ and $\gamma_{2,L}$ are given the same definitions as in Section III-A.

Space and Time Complexity. The space and time complexity of VS-1PASS can be derived in a similar manner as VS-2PASS. Both of them has the same space and time complexities. We omit the analysis details for the interest of space.

IV. Time-based Sliding Window

In the time-based sliding window model, edges expire at a fixed interval following their arrival. Denote the current time as T and the fixed expiration time as N, where N is also referred to as the window length. For this model, our goal is to compute an estimation of the number of butterflies in $G_{W^T_{T-N}}$, the graph consisting of all the current valid edges.

A. Basic solution

In this section, we first offer a basic method B-TSW, which deploys the priority sampling.

Substream strategy. Our algorithms employ substream-based methods, which partition the graph stream into multiple substreams for separate maintenance within the sliding window setting. This approach divides incoming edges from the graph stream into a predetermined number of substreams, denoted as k, by using a hash function $H : E \rightarrow \{1, 2, \cdots, k\}$. Each substream allocates fixed memory to store no more than one sampled edge. Each incoming edge is assigned a randomized priority value, where the randomized priority value is calculated by a hash function $\Pr : E \rightarrow (0, 1)$. In each substream, the edge with the highest priority value in the current window is chosen as the sampled edge. The current window refers to the set of all the current valid edges, denoted as W_c, which is W^T_{T-N} for $T \ge N$ and W^T_0 for $T < N$. These sampled edges form the current sampled graph, facilitating the estimated computations for the window in question at any time during the receiving process of the graph stream.

Blind areas. The blind area problem refers to the following situation. When a sampled edge in a substream s_i expires, the priority values of edges in s_i that arrive after the sampled edge (denote the edge set as E_b) are not tracked, forming the blind area. In the worst case, the substream cannot offer a valid sampled edge until all edges in E_b expire.

Inspired by the algorithm SWTC for triangle counting [11], we tackle the blind area problem with the slicing technique. In particular, it splits the timeline of the graph stream into slices of fixed size N, where N is the window size. Let T denote the current time point. Let T_{new} and T_{old} denote the latest and second latest splitting point no later than T, respectively. With

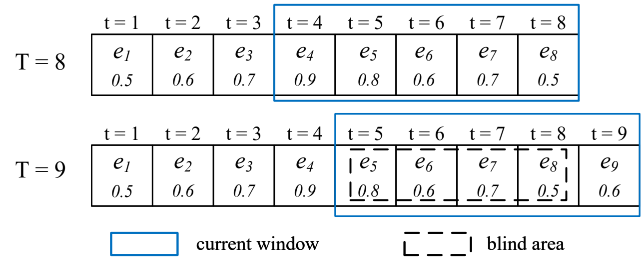


Fig. 2. Example of the blind area

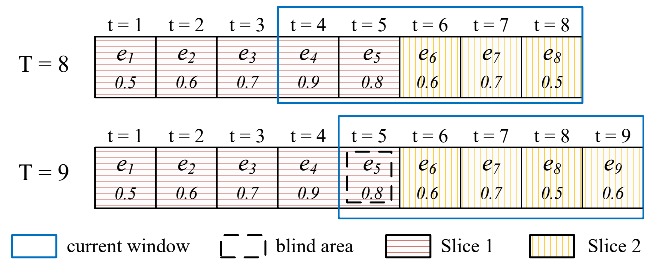


Fig. 3. Example of the slicing technique

the timeline split into slices of fixed size, the current window overlaps at most two slices, denoted as $W^{T_{new}}_{T_{old}}$ and $W^T_{T_{new}}$, respectively. For each of the slices, the edge of the highest priority value is recorded; hence, substreams can recover faster from the expiration of the previous current edges.

Example 2. *Here we offer a specific example of the blind area and the slicing technique with window size $N = 5$. As shown in Figure 2, when the current time $T = 8$, e_4 is the sample edge of substream s_i. When e_4 expires, i.e., $T = 9$, e_5, e_6, e_7, e_8 fall in the blind area because their priority values are not tracked before they are discarded. In the worst case, before all edges in the blind area expire, s_i can not offer a valid sample edge. With the timeline split into slices of fixed size, the current window overlaps at most two slices, as shown in Figure 3. When the window overlaps two slices, denote the sets of received edges within the two slices as $W^{T_{new}}_{T_{old}}$ and $W^T_{T_{new}}$. In this example, $T_{old} = 0$, and $T_{new} = 5$ according to its definition. In substream s_i, for each of the slices, the edge of the highest priority value is recorded, denoted as $\zeta^{T_{new}}_{T_{old}}[i]$ and $\zeta^T_{T_{new}}[i]$ respectively. Given the same situation, when the sample edge $\zeta^{T_{new}}_{T_{old}}[i] = e_4$ expires, the edge $\zeta^T_{T_{new}}[i] = e_7$ with the highest priority in $W^T_{T_{new}} = \{e_6, e_7, e_8\}$ is recorded. Therefore, the blind area shrinks to only $\{e_5\}$, and s_i can offer a valid sample edge no later than e_5 expires.*

Algorithm 4 shows the pseudo-code of the processing steps of our B-TSW algorithm for a newly arrived edge. $G^{(s)}$ denotes the sampled graph. Each of the edges entering the substreams is given two hash values. $H(e)$ determines which substream the edge e is allocated to (Line 1) and $\Pr(e)$ assigns a priority value to e. When the window overlaps two slices, denote the sets of received edges within the two

Algorithm 4: ProcessEdge - B-TSW $(e = (u, v))$

Input: Edge $e = (u, v)$
Output: Updated sample

1 $pos \leftarrow H(e)$;
2 **if** $\zeta^{T}_{T_{new}}[pos] = NULL\ or\ \Pr(\zeta^{T}_{T_{new}}[pos]) \leq \Pr(e)$ **then**
3   **if** $\zeta^{T}_{T_{new}}[pos] \neq NULL$ **then**
4         $G^{(s)}.delete(\zeta^{T}_{T_{new}}[pos])$;
5   **end**
6     $\zeta^{T}_{T_{new}}[p] \leftarrow e$;
7   **if** $\zeta^{T_{new}}_{T_{old}}[pos] = NULL\ or\ \Pr(\zeta^{T_{new}}_{T_{old}}[pos]) \leq \Pr(e)$ **then**
8         $G^{(s)}.add(e)$;
9   **if** $\zeta^{T_{new}}_{T_{old}}[p]\ is\ in\ G^{(s)}$ **then**
10             $G^{(s)}.delete(\zeta^{T_{new}}_{T_{old}}[p])$;
11   **end**
12   **end**
13 **end**

slices as $W^{T_{new}}_{T_{old}}$ and $W^{T}_{T_{new}}$. In substream s_i, for each of the slices, the edge of the highest priority value is recorded, denoted as $\zeta^{T_{new}}_{T_{old}}[i]$ and $\zeta^{T}_{T_{new}}[i]$ respectively. In the designated substream s_{pos}, $\Pr(e)$ is first compared with $\Pr(\zeta^{T}_{T_{new}}[pos])$ to maintain the record of the edge with the highest priority in $W^{T}_{T_{new}}$ (Line 2). Edge e is recorded as the edge with the highest priority in $W^{T}_{T_{new}}$ if $\Pr(e)$ is higher than or equal to $\zeta^{T}_{T_{new}}[pos]$ and discarded otherwise (Lines 3 to 6). Then, $\Pr(e)$ is compared with $\Pr(\zeta^{T_{new}}_{T_{old}}[pos])$ to decide whether e should be add to $G^{(s)}$ (Lines 7 to 12). If $\zeta^{T_{new}}_{T_{old}}[pos]$ is empty, two situations are possible: (i)$W^{T}_{T_{new}}$ is within the first slice; (ii) the edge with the highest priority in the last slice is already replaced by an edge e_j in $W^{T}_{T_{new}}$. For the first situation, e is obviously the highest prioritized edge in the current window. For the second situation, as e is the highest prioritized edge in $W^{T}_{T_{new}}$, $\Pr(e) \geq \Pr(e_j)$; hence, e has the highest priority in the current window. Therefore, when $\zeta^{T_{new}}_{T_{old}}[pos]$ is empty, e should be set as a valid sampled edge and added to $G^{(s)}$. If $\Pr(\zeta^{T_{new}}_{T_{old}}[pos]) \leq \Pr(e)$, e has a higher priority than any edge in $W^{T_{new}}_{T_{old}}$. Meanwhile, e is the edge with the highest priority in $W^{T}_{T_{new}}$. Therefore, e is the highest prioritized edge in the current window and should be added to $G^{(s)}$. To always remove the expired edge in the sampled graph $G^{(s)}$ in time, we check and delete all the expired edges in $G^{(s)}$ at every time unit. To achieve this efficiently, we maintain a time-sequence-based linked list for the sampled edges. At every time unit, we traverse the sampled edges from the oldest edge to the first edge that is valid and delete all the traversed edges except the last one.

Algorithm 5 shows the steps to compute the estimation of $\bowtie_{W_c}$. Let ω denote the valid sample size. Firstly, $|W^{T}_{T_{old}}|$ is estimated as $|\widehat{W^{T}_{T_{old}}}| = \frac{\alpha_k k^2}{\sum_{i=1}^{k} 2^{-R[i]}}$, where $\alpha_k = 0.7213/\left(1 + 1.079/k\right)$ and $R[1 \cdots k]$ is an array of variables maintained for each of the substreams to form a HyperLogLog sketch [10] (We have included a brief review of HyperLogLog sketch in Section II). For the i-th substream s_i, let θ_i denote the larger one between $\Pr\left(\zeta^{T}_{T_{new}}[i]\right)$ and $\Pr\left(\zeta^{T_{new}}_{T_{old}}[i]\right)$. $R[i]$ is computed as $\lceil -log\left(1 - \theta_i\right)\rceil$, 0 for an empty substream. We denote the number of edges in G_{W_c} as ϕ. The estimation of the number of edges in G_{W_c}, denoted as $\hat{\phi}$ can be computed as $|\widehat{W^{T}_{T_{old}}}| \cdot \frac{\omega}{k}$. We use the $ExactBFC$ algorithm [42] to compute the exact number of butterflies in the sampled graph. We obtain the estimation of $\bowtie_{G_{W_c}}$ as

Algorithm 5: ComputeEstimation - B-TSW

Input: The sampled graph $G^{(s)}$, valid sample size ω, number of substreams k, array $R[1 \cdots k]$
Output: Estimation of the number of butterflies in $G_{W^{T}_{T-N}}$

$|\widehat{W^{T}_{T_{old}}}| \leftarrow \frac{\alpha_k k^2}{\sum_{i=1}^{k} 2^{-R[i]}}$;
$\hat{\phi} \leftarrow |\widehat{W^{T}_{T_{old}}}| \cdot \frac{\omega}{k}$;
$\bowtie^{(s)} \leftarrow ExactBFC(G^{(s)})$;
$Y_s \leftarrow \bowtie^{(s)} \cdot \frac{\hat{\phi}(\hat{\phi}-1)(\hat{\phi}-2)(\hat{\phi}-3)}{\omega(\omega-1)(\omega-2)(\omega-3)}$

$$Y_s = \bowtie^{(s)} \cdot \frac{\hat{\phi}\left(\hat{\phi} - 1\right)\left(\hat{\phi} - 2\right)\left(\hat{\phi} - 3\right)}{\omega\left(\omega - 1\right)\left(\omega - 2\right)\left(\omega - 3\right)}.$$

Next, we derive the memory cost of our algorithm B-TSW, and the probability that a butterfly can be sampled.

Lemma 9. *The memory cost of B-TSW is bounded by $k \cdot m_e$, where m_e is the memory to keep all information of two edges in a substream.*

Lemma 10. *B-TSW samples each butterfly in G_{W_c} with probability:*

$$P_{\bowtie_s} = \frac{\omega\left(\omega - 1\right)\left(\omega - 2\right)\left(\omega - 3\right)}{\phi\left(\phi - 1\right)\left(\phi - 2\right)\left(\phi - 3\right)}.$$

Lemma 11. *For the current window W_c, the expectation of the estimation of B-TSW is:*

$$\mathbb{E}[Y_s] = \bowtie_{G_{W_c}}.$$

Let δ_0 denote the number of pair of butterflies in G_{W_c} that share no edge; δ_1 denotes the number of pair of butterflies in G_{W_c} that share one edge, δ_2 denote the number of pair of butterflies in G_{W_c} that share two edges.

Lemma 12. *For the window W_c, the variance of the estimation is*

$$\mathbb{V}\mathrm{ar}[Y_s] = \bowtie_{G_{W_c}}\left(\frac{1}{\theta_4} - 1\right) + \delta_0\left(\frac{\theta_8}{\theta_4^2} - 1\right) + \delta_1\left(\frac{\theta_7}{\theta_4^2} - 1\right) + \delta_2\left(\frac{\theta_6}{\theta_4^2} - 1\right).$$

Algorithm 6: ProcessEdge - VS-TSW $(e = (u, v), p)$

Input: Edge $e = (u, v)$, vertex sampling probability p
Output: Updated sample

```
1 if hash(u) ≤ p then
2 |   Execute the processing steps as in B-TSW;
3 end
```

Algorithm 7: ComputeEstimation - VS-TSW

Input: The sampled graph $G^{(s)}$, valid sample size ω, number of substreams k, array $R[1 \cdots k]$
Output: Estimation of the number of butterflies in $G_{W_{T-N}^{T}}$

1. $\widehat{|W_{T_{old}}^{T}|} \leftarrow \frac{\alpha_k k^2}{\sum_{i=1}^{k} 2^{-R[i]}}$;
2. $\hat{\phi} \leftarrow \widehat{|W_{T_{old}}^{T}|} \cdot \frac{\omega}{k}$;
3. $Y_v = \bowtie^{(s)} \cdot \frac{\hat{\phi}(p\hat{\phi}-1)(p\hat{\phi}-2)(p\hat{\phi}-3c)}{p\omega(\omega-1)(\omega-2)(\omega-3)}$

B. Improved solution

In this section, we propose our improved version of B-TSW algorithm, denoted as *VS-TSW*, which integrate the vertex sparsification technique with the basic solution B-TSW.

Algorithms 6–7 shows the pseudo-code of our method VS-TSW. We have a sampling probability p for sampling vertices on one side of the graph. Without loss of generality, let the sampled side be L. After receiving an edge $e = (u, v)(u \in L, v \in R)$ from the graph stream, we apply the hash function $hash : V \rightarrow (0, 1)$ on u and use the hash value to decide whether to discard e. If $hash(u) > p$, e is discarded. If u is not discarded, the same processing steps are executed as in B-TSW (Line 2). We denote the number of butterflies in the sampled graph as $\bowtie^{(s)}$. Then, we can estimate the number of butterflies in $G_{W_{T-N}^{T}}$ as

$$Y_v = \bowtie^{(s)} \cdot \frac{\hat{\phi}\left(p\hat{\phi}-1\right)\left(p\hat{\phi}-2\right)\left(p\hat{\phi}-3c\right)}{p\omega\left(\omega-1\right)\left(\omega-2\right)\left(\omega-3\right)}.$$

Lemma 13. *Given the same window length and number of substreams, VS-TSW samples each butterfly in G_{W_c} with probability*

$$P_{\bowtie_v} = \frac{p\omega\left(\omega-1\right)\left(\omega-2\right)\left(\omega-3\right)}{\phi\left(p\phi-1\right)\left(p\phi-2\right)\left(p\phi-3\right)} \geq \frac{P_{\bowtie_s}}{p^2}.$$

We show that the estimation of VS-TSW is unbiased and analyze the variance of the estimation. The proofs are listed in our technical report.

Lemma 14. *For the current window W_c, the expectation of the estimation of VS-TSW is:*

$$\mathbb{E}[Y_v] = \bowtie_{G_{W_c}}$$

Let $\eta_{i,j}$ denote the number of unordered pairs of butterflies in G_{W_c} that share i vertices on the L side and j edges. Let

$$\tau_i = \frac{\omega\left(\omega-1\right)\left(\omega-2\right)\cdots\left(\omega-i\right)}{p\phi\left(p\phi-1\right)\left(p\phi-2\right)\cdots\left(p\phi-i\right)}$$

Lemma 15. *For the current window W_c, the variance of the estimation of VS-TSW is:*

$$\begin{aligned}
\mathbb{V}\mathrm{ar}[Y_v] = &\bowtie_{G_{W_c}}\left(\frac{1}{p^2\tau_4}-1\right) \\
&+ 2\eta_{0,0}\left(\frac{\tau_8}{\tau_4^2}-1\right) + 2\eta_{1,0}\left(\frac{\tau_8}{p\tau_4^2}-1\right) \\
&+ 2\eta_{1,1}\left(\frac{\tau_7}{p\tau_4^2}-1\right) + 2\eta_{1,2}\left(\frac{\tau_6}{p\tau_4^2}-1\right) \\
&+ 2\eta_{2,0}\left(\frac{\tau_8}{p^2\tau_4^2}-1\right) + 2\eta_{2,2}\left(\frac{\tau_6}{p^2\tau_4^2}-1\right)
\end{aligned}$$

V. Theoretical Analysis

In this section, we provide missing proofs of the lemmas in Section III and Section IV.

A. Proof of Lemma 3

For a random butterfly $B = \{u_1, u_2, v_1, v_2\}$ $(u_1, u_2 \in L, v_1, v_2 \in R)$, B is sampled when both of its co-affiliations $\Lambda_1 = \{u_1, v_1, u_2\}$ and $\Lambda_2 = \{u_1, v_2, u_2\}$ are sampled. When u_1 and u_2 are both sampled or neither sampled in the first path, neither of Λ_1 and Λ_2 are sampled. When only one of u_1 and u_2 is sampled in the first pass, both Λ_1 and Λ_2 are sampled. Therefore, the probability of sampling a butterfly B is the probability of sampling exactly one of its two vertices on the L side, that is

$$\begin{aligned}
P_{\bowtie_2} &= 1 - P\left(hash\left(u_1\right) \leq p \wedge hash\left(u_2\right) \leq p\right) \\
&\quad - P\left(hash\left(u_1\right) > p \wedge hash\left(u_2\right) > p\right) \\
&= 1 - p^2 - \left(1-p\right)^2 \\
&= 2p - 2p^2
\end{aligned}$$

B. Proof of Lemma 4

Let $\bowtie^{(s)}$ denote the number of sampled butterflies. We have

$$Y_2 = \frac{F_L - c_L}{4p - 4p^2} = \frac{F_L - c_L}{2}\frac{1}{2p-2p^2} = \bowtie^{(s)}\frac{1}{2p-2p^2}$$

According to Lemma 3, VCA-2PASS samples a butterfly with probability $P_{\bowtie_2} = 2p - 2p^2$. Label each butterfly in S with indexes from 1 to $\bowtie_S$. For the i-th butterfly B_i, let σ_i be the random variable that equals 1 when B_i is sampled, and 0 otherwise. Then we have,

$$\begin{aligned}
\mathbb{E}[Y_2] &= \mathbb{E}[\bowtie^{(s)}\frac{1}{2p-2p^2}] = \frac{1}{2p-2p^2}\mathbb{E}[\sum_{i=1}^{\bowtie_S}\sigma_i] \\
&= \frac{1}{2p-2p^2}\sum_{i=1}^{\bowtie_S}\mathbb{E}[\sigma_i] = \frac{1}{2p-2p^2}\bowtie_S P_{\bowtie_2} \\
&= \frac{1}{2p-2p^2}\bowtie_S\left(2p-2p^2\right) = \bowtie_S
\end{aligned}$$

C. Proof of Lemma 5

Denote the butterfly count in S as $\bowtie_S$ and the probability of sampling a butterfly as $P_{\bowtie}$. We have

$$\begin{aligned}\mathbb{V}\mathrm{ar}[Y_2] &= \frac{\sum_{i=1}^{\bowtie_S} \mathbb{V}\mathrm{ar}[\sigma_i] + \sum_{i\neq j} cov\left(\sigma_i,\sigma_j\right)}{P_{\bowtie}^2} \\ &= \frac{\bowtie_S\left(P_{\bowtie} - P_{\bowtie}^2\right) + \sum_{i\neq j} cov\left(\sigma_i,\sigma_j\right)}{P_{\bowtie}^2} \\ &= \bowtie_S\left(\frac{1}{P_{\bowtie}} - 1\right) + \frac{\sum_{i\neq j} cov\left(\sigma_i,\sigma_j\right)}{P_{\bowtie}^2} \\ &= \bowtie_S\left(\frac{1}{2p-2p^2} - 1\right) + \frac{\sum_{i\neq j} cov\left(\sigma_i,\sigma_j\right)}{\left(2p-2p^2\right)^2}\end{aligned}$$

To compute $\sum_{i\neq j} cov\left(\sigma_i,\sigma_j\right)$, we need to consider situations where the sampling of two butterflies is correlated. Label each butterfly in S and set variables σ_i the same as in the proof of Lemma 4. The situations are listed as follows:

1) Two butterflies share one vertex on the L side. Given two butterflies $B_i = \{u_1,u_2,v_1,v_2\}$ and $B_j = \{u_2,u_3,v_2,v_3\}$ ($u_i \in L, v_i \in R, i = 1,2,3$) that share the vertex u_2 on L side. The circumstance where B_i and B_j are both sampled is when u_2 is sampled while u_1 and u_3 are not sampled, or u_1 and u_3 are sampled while u_2 is not sampled. The probability of B_i and B_j both being sampled can be computed as

$$P\left(\sigma_i\sigma_j = 1\right) = p\left(1-p\right)^2 + p^2\left(1-p\right) = p - p^2.$$

$cov\left(\sigma_i,\sigma_j\right)$ can be computed as:

$$\begin{aligned}cov\left(\sigma_i,\sigma_j\right) &= \mathbb{E}[\sigma_i\sigma_j] - \mathbb{E}[\sigma_i]\mathbb{E}[\sigma_j] \\ &= \left(p-p^2\right) - \left(2p-2p^2\right)^2.\end{aligned}$$

Each of such butterfly pairs accounts for two terms in $\sum_{i\neq j} cov\left(\sigma_i,\sigma_j\right)$: $cov\left(\sigma_i,\sigma_j\right)$ and $cov\left(\sigma_j,\sigma_i\right)$.

2) Two butterflies share two vertices on the L side. Given two butterflies $B_i = \{u_1,u_2,v_1,v_2\}$ and $B_j = \{u_1,u_2,v_2,v_3\}$ ($u_i \in L, v_i \in R, i = 1,2,3$) that share the vertex u_2 on L side. This subgraph contains another butterfly $B_k = \{u_1,u_2,v_1,v_3\}$. The circumstance where B_i and B_j are both sampled is when u_1 is sampled while u_2 is not sampled, or u_2 is sampled while u_1 is not sampled. The probability of B_i and B_j both being sampled can be computed as

$$P\left(\sigma_i\sigma_j = 1\right) = 1 - p^2 - \left(1-p\right)^2 = 2p - 2p^2.$$

$cov\left(\sigma_i,\sigma_j\right)$ can be computed as:

$$\begin{aligned}cov\left(\sigma_i,\sigma_j\right) &= \mathbb{E}[\sigma_i\sigma_j] - \mathbb{E}[\sigma_i]\mathbb{E}[\sigma_j] \\ &= \left(2p-2p^2\right) - \left(2p-2p^2\right)^2.\end{aligned}$$

Each of such butterfly pairs accounts for two terms in $\sum_{i\neq j} cov\left(\sigma_i,\sigma_j\right)$.

Therefore, $\mathbb{V}\mathrm{ar}[Y_2]$ can be computed as

$$\begin{aligned}\mathbb{V}\mathrm{ar}[Y_2] &= \bowtie_S\left(\frac{1}{2p-2p^2} - 1\right) \\ &+ \frac{2\gamma_{1,L}\left(\left(p-p^2\right) - \left(2p-2p^2\right)^2\right) + 2\gamma_{2,L}\left(\left(2p-2p^2\right) - \left(2p-2p^2\right)\cdots\right.}{\left(2p-2p^2\right)^2} \\ &= \bowtie_S\left(\frac{1}{2p-2p^2} - 1\right) + 2\gamma_{1,L}\left(\frac{1}{2\left(2p-2p^2\right)} - 1\right) + 2\gamma_{2,L}\left(\frac{1}{2p-2\cdots}\right. \\ &= \bowtie_S\left(\frac{1}{2p-2p^2} - 1\right) + \frac{\gamma_{1,L} + 2\gamma_{2,L}}{2p-2p^2} - \left(2\gamma_{1,L} + 2\gamma_{2,L}\right).\end{aligned}$$

D. Proof of Lemma 6

For a random butterfly $B = \{u_1,u_2,v_1,v_2\}$ ($u_1,u_2 \in L, v_1,v_2 \in R$),we label edges $\left(u_1,v_1\right),\left(u_1,v_2\right),\left(u_2,v_1\right),\left(u_2,v_2\right)$ as e_1,e_2,e_3,e_4. We consider the following situations (let $P_{\bowtie}^{(i)}$ denote the probability of sampling a butterfly in the i-th situation):

1) Edges e_1,e_2 arrives before e_3,e_4. The probability of this situation is $\frac{1}{6}$. B is sampled if and only if u_1 is sampled. Therefore, $P_{\bowtie}^{(1)} = p$.
2) Edges e_2,e_4 arrives before e_1,e_2. The probability of this situation is $\frac{1}{6}$. B is sampled if and only if u_2 is sampled. Therefore, $P_{\bowtie}^{(2)} = p$.
3) Edges arrive in sequence e_1,e_3,e_2,e_4 or e_2,e_4,e_1,e_3. The probability of this situation is $\frac{1}{12}$. B is sampled if and only if u_1 is sampled. Therefore, $P_{\bowtie}^{(3)} = p$.
4) Edges arrive in sequence e_3,e_1,e_4,e_2 or e_4,e_2,e_3,e_1. The probability of this situation is $\frac{1}{12}$. B is sampled if and only if u_2 is sampled. Therefore, $P_{\bowtie}^{(4)} = p$.
5) Edges e_1,e_4 arrives before e_2,e_3 or Edges e_2,e_3 arrives before e_1,e_4. The probability of this situation is $\frac{1}{3}$. B is sampled if and only if both u_1 and u_2 are sampled. Therefore, $P_{\bowtie}^{(5)} = p^2$.
6) Between edges e_3 and e_4, one arrives first among the four edges, and the other arrives the last. The probability of this situation is $\frac{1}{6}$. B is sampled if and only if both u_1 and u_2 are sampled. Therefore, $P_{\bowtie}^{(6)} = p^2$.

Therefore, let P_i denote the probability of the i-th situation, we have

$$P_{\bowtie 1} = \sum_{i=1}^{6} P_i P_{\bowtie}^{(i)} = \frac{p+p^2}{2}$$

E. Proof of Lemma 7

Let $\bowtie^{(s)}$ denote the number of sampled butterflies. We have

$$Y_1 = \frac{F_L - c_L}{p+p^2} = \frac{F_L - c_L}{2}\frac{2}{p+p^2} = \bowtie^{(s)}\frac{2}{p+p^2}$$

According to Lemma 6, VCA-1PASS samples a butterfly with probability $P_{\bowtie 1} = \frac{p+p^2}{2}$. Label each butterfly in S and set variables σ_i the same as in the proof of Lemma 4. Then we have,

$$
\begin{aligned}
\mathbb{E}[Y_1] &= \mathbb{E}[\bowtie^{(s)}\frac{2}{p+p^2}] = \frac{2}{p+p^2}\mathbb{E}[\sum_{i=1}^{\bowtie_S}\sigma_i] \\
&= \frac{2}{p+p^2}\sum_{i=1}^{\bowtie_S}\mathbb{E}[\sigma_i] = \frac{2}{p+p^2}\bowtie_S P_{\bowtie 1} \\
&= \frac{2}{p+p^2}\bowtie_S\left(\frac{p+p^2}{2}\right) = \bowtie_S
\end{aligned}
$$

F. Proof of Lemma 8

Denote the butterfly count in S as $\bowtie_S$ and the probability of sampling a butterfly as $P_{\bowtie}$. We have

$$
\begin{aligned}
\mathbb{V}\text{ar}[Y_1] &= \frac{\sum_{i=1}^{\bowtie_S}\mathbb{V}\text{ar}[\sigma_i] + \sum_{i\neq j} cov\left(\sigma_i,\sigma_j\right)}{P_{\bowtie}^2} \\
&= \frac{\bowtie_S\left(P_{\bowtie} - P_{\bowtie}^2\right) + \sum_{i\neq j} cov\left(\sigma_i,\sigma_j\right)}{P_{\bowtie}^2} \\
&= \bowtie_S\left(\frac{1}{P_{\bowtie}} - 1\right) + \frac{\sum_{i\neq j} cov\left(\sigma_i,\sigma_j\right)}{P_{\bowtie}^2} \\
&= \bowtie_S\left(\frac{2}{p+p^2} - 1\right) + \frac{\sum_{i\neq j} cov\left(\sigma_i,\sigma_j\right)}{\left(\frac{p+p^2}{2}\right)^2}
\end{aligned}
$$

To compute $\sum_{i\neq j} cov\left(\sigma_i,\sigma_j\right)$, we consider the situations where the sampling of two butterflies is correlated. The situations are listed as follows:

1) Two butterflies share one vertex on the L side. Given two butterflies $B_i = \{u_1,u_2,v_1,v_2\}$ and $B_j = \{u_2,u_3,v_2,v_3\}$ $(u_i \in L, v_i \in R, i = 1,2,3)$ that share the vertex u_2 on L side. The probability of sampling both butterflies is less than sampling one butterfly. We have

$$P\left(\sigma_i\sigma_j = 1\right) \leq P_{\bowtie 1} = \frac{p+p^2}{2}$$

Therefore,

$$
\begin{aligned}
cov\left(\sigma_i,\sigma_j\right) &= \mathbb{E}[\sigma_i\sigma_j] - \mathbb{E}[\sigma_i]\mathbb{E}[\sigma_j] \\
&\leq \frac{p+p^2}{2} - \left(\frac{p+p^2}{2}\right)^2
\end{aligned}
$$

Each of such butterfly pairs accounts for two terms in $\sum_{i\neq j} cov\left(\sigma_i,\sigma_j\right)$: $cov\left(\sigma_i,\sigma_j\right)$ and $cov\left(\sigma_j,\sigma_i\right)$.

2) Two butterflies share two vertices on the L side. Given two butterflies $B_i = \{u_1,u_2,v_1,v_2\}$ and $B_j = \{u_1,u_2,v_2,v_3\}$ $(u_i \in L, v_i \in R, i = 1,2,3)$ that share the vertex u_2 on L side. This subgraph contains another butterfly $B_k = \{u_1,u_2,v_1,v_3\}$. Similar to the first situation, we have

$$cov\left(\sigma_i,\sigma_j\right) \leq \frac{p+p^2}{2} - \left(\frac{p+p^2}{2}\right)^2$$

Each of such butterfly pairs accounts for two terms in $\sum_{i\neq j} cov\left(\sigma_i,\sigma_j\right)$.

Therefore, we have

$$
\begin{aligned}
\mathbb{V}\text{ar}[Y_1] &\leq \bowtie_S\left(\frac{2}{p+p^2} - 1\right) \\
&\quad + \frac{2\gamma_{1,L}\left(\frac{p+p^2}{2} - \left(\frac{p+p^2}{2}\right)^2\right) + 2\gamma_{2,L}\left(\frac{p+p^2}{2} - \left(\frac{p+p^2}{2}\right)^2\right)}{\left(\frac{p+p^2}{2}\right)^2} \\
&= \bowtie_S\left(\frac{2}{p+p^2} - 1\right) + \frac{4\gamma_{1,L} + 4\gamma_{2,L}}{p+p^2} - \left(2\gamma_{1,L} + 2\gamma_{2,L}\right)
\end{aligned}
$$

G. Proof of Lemma 9

In SS-TSW, the memory usage is dominated by the memory used to maintain the sampled graph. Two edges are stored for a sub-stream at most. Let m_e denote the memory for storing all the information of two edges. Then, we have that the memory usage of SS-TSW is upper-bounded by km_e.

H. Proof of Lemma 10

Given a random butterfly $B_i = \{u_1,u_2,v_1,v_2\}$ in G_{W_c}, the probability of B_i being sampled is the probability of the four edges in B all being sampled. Because each edge is sampled with equal probability, each combination of sampled edges appears with the same probability. Randomly selecting ω sampled edges from ϕ edges in G_{W_c}, there are $\binom{\phi}{\omega}$ combinations. For the combinations containing the four edges in B_i, we have $\binom{\phi-4}{\omega-4}$ combinations. Therefore, B_i is sampled with probability

$$P_{\bowtie s} = \binom{\phi}{\omega}\cdot\binom{\phi-4}{\omega-4} = \frac{\omega\left(\omega-1\right)\left(\omega-2\right)\left(\omega-3\right)}{\phi\left(\phi-1\right)\left(\phi-2\right)\left(\phi-3\right)}$$

I. Proof of Lemma 11

According to Lemma 10, each butterfly is sampled with probability $P_{\bowtie s} = \frac{\omega(\omega-1)(\omega-2)(\omega-3)}{\phi(\phi-1)(\phi-2)(\phi-3)}$. Label each butterfly in S with indexes from 1 to b, $b = \bowtie_{G_{W_c}}$. For the i-th butterfly B_i, let σ_i be the random variable that equals 1 when B_i is sampled, and 0 otherwise. We have $\mathbb{E}[\sigma_i] = P_{\bowtie s} = \frac{\omega(\omega-1)(\omega-2)(\omega-3)}{\phi(\phi-1)(\phi-2)(\phi-3)}$. Therefore, we have

$$
\begin{aligned}
\mathbb{E}[Y_s] &= \mathbb{E}[\sum_{i=1}^{b}\sigma_i\frac{\hat{\phi}\left(\hat{\phi}-1\right)\left(\hat{\phi}-2\right)\left(\hat{\phi}-3\right)}{\omega\left(\omega-1\right)\left(\omega-2\right)\left(\omega-3\right)}] \\
&= \sum_{i=1}^{b}\mathbb{E}[\sigma_i\frac{\hat{\phi}\left(\hat{\phi}-1\right)\left(\hat{\phi}-2\right)\left(\hat{\phi}-3\right)}{\omega\left(\omega-1\right)\left(\omega-2\right)\left(\omega-3\right)}]
\end{aligned}
$$

Note that with the same ϕ, $\hat{\phi}$ is not related to σ_i and is an unbiased estimation of ϕ So we have

$$
\begin{aligned}
\mathbb{E}[Y_s] &= \sum_{i=1}^{b}\mathbb{E}[\sigma_i]\cdot\mathbb{E}[\sigma_i]\mathbb{E}[\frac{\hat{\phi}\left(\hat{\phi}-1\right)\left(\hat{\phi}-2\right)\left(\hat{\phi}-3\right)}{\omega\left(\omega-1\right)\left(\omega-2\right)\left(\omega-3\right)}] \\
&= \frac{\phi\left(\phi-1\right)\left(\phi-2\right)\left(\phi-3\right)}{\omega\left(\omega-1\right)\left(\omega-2\right)\left(\omega-3\right)}\cdot\sum_{i=1}^{b}\mathbb{E}[\sigma_i] \\
&= \frac{\phi\left(\phi-1\right)\left(\phi-2\right)\left(\phi-3\right)}{\omega\left(\omega-1\right)\left(\omega-2\right)\left(\omega-3\right)}\cdot\frac{\omega\left(\omega-1\right)\left(\omega-2\right)\left(\omega-3\right)}{\phi\left(\phi-1\right)\left(\phi-2\right)\left(\phi-3\right)}\cdot b \\
&= b = \bowtie_{G_{W_c}}
\end{aligned}
$$

J. Proof of Lemma 12

Let $b = \bowtie_{G_{W_c}}$. We label the butterflies in G_{W_c} and set variables $\sigma_1, \cdots, \sigma_b$ as in the proof of Lemma 11. Denote the probability of sampling a butterfly as $P_{\bowtie s}$. We have

$$
\begin{aligned}
\mathbb{V}\mathrm{ar}[Y_s] &= \frac{\sum_{i=1}^{b} \mathbb{V}\mathrm{ar}[\sigma_i] + \sum_{i \neq j} cov\left(\sigma_i, \sigma_j\right)}{P_{\bowtie s}^2} \\
&= \frac{b\left(P_{\bowtie s} - P_{\bowtie s}^2\right) + \sum_{i \neq j} cov\left(\sigma_i, \sigma_j\right)}{P_{\bowtie s}^2} \\
&= b\left(\frac{1}{P_{\bowtie s}} - 1\right) + \frac{\sum_{i \neq j} cov\left(\sigma_i, \sigma_j\right)}{P_{\bowtie s}^2}
\end{aligned}
$$

To compute $\sum_{i \neq j} cov\left(\sigma_i, \sigma_j\right)$, we consider situations where the sampling of two butterflies is correlated. Let $\theta_i = \frac{\omega(\omega-1)(\omega-2)\cdots(\omega-i+1)}{\phi(\phi-1)(\phi-2)\cdots(\phi-i+1)}$. The situations are listed as follows.

1) Two butterflies share no edge. If two butterflies share no edge, the probability of sampling both butterflies is the probability of sampling all eight edges in the two butterflies, which can be computed as

$$P\left(\sigma_i \sigma_j = 1\right) = \theta_8$$

$cov\left(\sigma_i, \sigma_j\right)$ can be computed as

$$
\begin{aligned}
cov\left(\sigma_i, \sigma_j\right) &= \mathbb{E}[\sigma_i \sigma_j] - \mathbb{E}[\sigma_i]\mathbb{E}[\sigma_j] \\
&= \theta_8 - \theta_4^2.
\end{aligned}
$$

2) Two butterflies share one edge. Given two butterflies $B_i = \{u_1, u_2, v_1, v_2\}$ and $B_j = \{u_2, u_3, v_2, v_3\}$ that share an edge (u_2, v_2). The probability of sampling both B_i and B_j is the probability of sampling all seven edges in the two butterflies, which can be computed as

$$P\left(\sigma_i \sigma_j = 1\right) = \theta_7$$

$cov\left(\sigma_i, \sigma_j\right)$ can be computed as

$$
\begin{aligned}
cov\left(\sigma_i, \sigma_j\right) &= \mathbb{E}[\sigma_i \sigma_j] - \mathbb{E}[\sigma_i]\mathbb{E}[\sigma_j] \\
&= \theta_7 - \theta_4^2.
\end{aligned}
$$

each of such relation accounts for two terms in $\sum_{i \neq j} cov\left(\sigma_i, \sigma_j\right)$.

3) Two butterflies share two edges. Given two butterflies $B_i = \{u_1, u_2, v_1, v_2\}$ and $B_j = \{u_2, u_3, v_1, v_2\}$ share two edges (u_2, v_1) and (u_2, v_2). The probability of sampling both B_i and B_j is the probability of sampling all six edges in the two butterflies, which can be computed as

$$P\left(\sigma_i \sigma_j = 1\right) = \theta_6$$

$cov\left(\sigma_i, \sigma_j\right)$ can be computed as

$$
\begin{aligned}
cov\left(\sigma_i, \sigma_j\right) &= \mathbb{E}[\sigma_i \sigma_j] - \mathbb{E}[\sigma_i]\mathbb{E}[\sigma_j] \\
&= \theta_6 - \theta_4^2.
\end{aligned}
$$

Therefore, the variance of estimation Y_s can be computed as

$$
\begin{aligned}
Var[Y_s] = b&\left(\frac{1}{P_{\bowtie s}} - 1\right) \\
&+ \frac{\delta_0\left(\theta_8 - \theta_4^2\right) + \delta_1\left(\theta_8 - \theta_4^2\right) + \delta_2\left(\theta_8 - \theta_4^2\right)}{P_{\bowtie s}^2} \\
&= \bowtie_{G_{W_c}}\left(\frac{1}{\theta_4} - 1\right) + \delta_0\left(\frac{\theta_8}{\theta_4^2} - 1\right) + \delta_1\left(\frac{\theta_7}{\theta_4^2} - 1\right) + \delta_2\left(\frac{\theta_6}{\theta_4^2} -\right.
\end{aligned}
$$

K. Proof of Lemma 13

Let $B = \{u_1, u_2, v_1, v_2\}\,(u_1, u_2 \in L,\ v_1, v_2 \in R)$ be a random butterfly in G_{W_c}.. For B to be sampled, both u_1 and u_2 need to be sampled, the probability of which is p^2. With each edge retained with probability p, the expectation of the number of retained edges in W_c is $p\phi$. Hence, if u_1 and u_2 are sampled, B is then sampled with probability $\frac{\omega(\omega-1)(\omega-2)(\omega-3)}{p\phi(p\phi-1)(p\phi-2)(p\phi-3)}$. Therefore, we have

$$
\begin{aligned}
P_{\bowtie c} &= p^2 \cdot \frac{\omega\left(\omega-1\right)\left(\omega-2\right)\left(\omega-3\right)}{p\phi\left(p\phi-1\right)\left(p\phi-2\right)\left(p\phi-3\right)} \\
&= \frac{p\omega\left(\omega-1\right)\left(\omega-2\right)\left(\omega-3\right)}{\phi\left(p\phi-1\right)\left(p\phi-2\right)\left(p\phi-3\right)} \\
&\geq p^2 \cdot \frac{\omega\left(\omega-1\right)\left(\omega-2\right)\left(\omega-3\right)}{p\phi p\left(\phi-1\right)p\left(\phi-2\right)p\left(\phi-3\right)} \\
&= \frac{p^2}{p^4} \cdot \frac{\omega\left(\omega-1\right)\left(\omega-2\right)\left(\omega-3\right)}{\phi\left(\phi-1\right)\left(\phi-2\right)\left(\phi-3\right)} \\
&= \frac{P_{\bowtie s}}{p^2}
\end{aligned}
$$

L. Proof of Lemma 14

According to Lemma 13, we have

$$P_{\bowtie v} = \frac{p\omega\left(\omega-1\right)\left(\omega-2\right)\left(\omega-3\right)}{\phi\left(p\phi-1\right)\left(p\phi-2\right)\left(p\phi-3\right)}$$

Similar to the proof of Lemma 11, we have

$$
\begin{aligned}
\mathbb{E}[Y_v] &= \frac{\phi\left(p\phi-1\right)\left(p\phi-2\right)\left(p\phi-3\right)}{p\omega\left(\omega-1\right)\left(\omega-2\right)\left(\omega-3\right)} \\
&\cdot \frac{p\omega\left(\omega-1\right)\left(\omega-2\right)\left(\omega-3\right)}{\phi\left(p\phi-1\right)\left(p\phi-2\right)\left(p\phi-3\right)} \\
&= \bowtie_{G_{W_c}}
\end{aligned}
$$

M. Proof of Lemma 15

Same as in the proof of Lemma 12, we consider all situations where the sampling of two butterflies is correlated to compute $\sum_{i \neq j} cov\left(\sigma_i, \sigma_j\right)$. In the examples, we assume vertices labeled as u_i are from the L side and v_i from the R side.

1) Two butterflies share no vertex in L or edge. Given two butterflies $B_i = \{u_1, u_2, v_1, v_2\}$ and $B_j = \{u_3, u_4, v_3, v_4\}$ $(u_i \in L, v_i \in R, i = 1, 2, 3, 4)$ that share no vertex in L or edge. The probability of all edges in B_i and B_j being retained is p^4. If all edges are retained,

the probability of B_i and B_j both being sampled is τ_8. Hence, we have

$$P(\sigma_i \sigma_j = 1) = p^3 \cdot \tau_7$$

$cov(\sigma_i, \sigma_j)$ can be computed as

$$\begin{aligned} cov(\sigma_i, \sigma_j) &= \mathbb{E}[\sigma_i \sigma_j] - \mathbb{E}[\sigma_i]\mathbb{E}[\sigma_j] \\ &= p^4 \tau_8 - p^4 \tau_4. \end{aligned}$$

2) Two butterflies share one vertex in L and no edge. Given two butterflies $B_i = \{u_1, u_2, v_1, v_2\}$ and $B_j = \{u_2, u_3, v_3, v_4\}$ $(u_i \in L, v_i \in R, i = 1, 2, 3, 4)$ that share one vertex u_2 in L. The probability of all edges in B_i and B_j being retained is p^3. If all edges enter the stream, the probability of B_i and B_j both being sampled is τ_8. Therefore, we have

$$\begin{aligned} cov(\sigma_i, \sigma_j) &= \mathbb{E}[\sigma_i \sigma_j] - \mathbb{E}[\sigma_i]\mathbb{E}[\sigma_j] \\ &= p^3 \tau_8 - p^4 \tau_4. \end{aligned}$$

3) Two butterflies share one vertex in L and one edge. Given two butterflies $B_i = \{u_1, u_2, v_1, v_2\}$ and $B_j = \{u_2, u_3, v_2, v_3\}$ $(u_i \in L, v_i \in R, i = 1, 2, 3)$ that share one vertex u_2 and one edge (u_2, v_2). The probability of all edges in B_i and B_j being retained is p^3. If all edges enter the stream, the probability of B_i and B_j both being sampled is τ_7. Therefore, we have

$$\begin{aligned} cov(\sigma_i, \sigma_j) &= \mathbb{E}[\sigma_i \sigma_j] - \mathbb{E}[\sigma_i]\mathbb{E}[\sigma_j] \\ &= p^3 \tau_7 - p^4 \tau_4. \end{aligned}$$

4) Two butterflies share one vertex in L and two edges. Given two butterflies $B_i = \{u_1, u_2, v_1, v_2\}$ and $B_j = \{u_2, u_3, v_1, v_2\}$ $(u_i \in L, v_i \in R, i = 1, 2, 3)$ that share one vertex u_2 in L two edges (u_2, v_1) and (u_2, v_2). The probability of all edges in B_i and B_j being retained is p^3. If all edges enter the stream, the probability of B_i and B_j both being sampled is τ_6. Therefore, we have

$$\begin{aligned} cov(\sigma_i, \sigma_j) &= \mathbb{E}[\sigma_i \sigma_j] - \mathbb{E}[\sigma_i]\mathbb{E}[\sigma_j] \\ &= p^3 \tau_6 - p^4 \tau_4. \end{aligned}$$

5) Two butterflies share two vertices and no edge. Given two butterflies $B_i = \{u_1, u_2, v_1, v_2\}$ and $B_j = \{u_1, u_2, v_3, v_4\}$ $(u_i \in L, v_i \in R, i = 1, 2, 3, 4)$ that share two vertices u_1 and u_2 and no edge. The probability of all edges in B_i and B_j being retained is p^2. If all edges enter the stream, the probability of B_i and B_j both being sampled is τ_8. Therefore, we have

$$\begin{aligned} cov(\sigma_i, \sigma_j) &= \mathbb{E}[\sigma_i \sigma_j] - \mathbb{E}[\sigma_i]\mathbb{E}[\sigma_j] \\ &= p^2 \tau_8 - p^4 \tau_4. \end{aligned}$$

6) Two butterflies share two vertices and two edges. Given two butterflies $B_i = \{u_1, u_2, v_1, v_2\}$ and $B_j = \{u_1, u_2, v_2, v_3\}$ $(u_i \in L, v_i \in R, i = 1, 2, 3)$ that share two vertices u_1 and u_2 and no edge. The probability of all edges in B_i and B_j being retained is p^2. If all edges enter the stream, the probability of B_i and B_j both being sampled is τ_6. Therefore, we have

$$\begin{aligned} cov(\sigma_i, \sigma_j) &= \mathbb{E}[\sigma_i \sigma_j] - \mathbb{E}[\sigma_i]\mathbb{E}[\sigma_j] \\ &= p^2 \tau_6 - p^4 \tau_4. \end{aligned}$$

Adding up all the situations, the variance of estimation Y_c can be computed as

$$\begin{aligned} Var[Y_c] = &\boxtimes_{G_{W_c}} \left(\frac{1}{p^2 \tau_4} - 1 \right) \\ &+ \frac{1}{p^4 \tau_4} \left(2\eta_{0,0} \left(p^4 \tau_8 - p^4 \tau_4 \right) + 2\eta_{1,0} \left(p^3 \tau_8 - p^4 \tau_4 \right) \right. \\ &+ 2\eta_{1,1} \left(p^3 \tau_7 - p^4 \tau_4 \right) + 2\eta_{1,2} \left(p^3 \tau_6 - p^4 \tau_4 \right) \\ &\left. + 2\eta_{2,0} \left(p^2 \tau_8 - p^4 \tau_4 \right) + 2\eta_{2,2} \left(p^2 \tau_6 - p^4 \tau_4 \right) \right) \\ &= \boxtimes_{G_{W_c}} \left(\frac{1}{p^2 \tau_4} - 1 \right) + 2\eta_{0,0} \left(\frac{\tau_8}{\tau_4^2} - 1 \right) + 2\eta_{1,0} \left(\frac{\tau_8}{p \tau_4^2} - 1 \right) \\ &+ 2\eta_{1,1} \left(\frac{\tau_7}{p \tau_4^2} - 1 \right) + 2\eta_{1,2} \left(\frac{\tau_6}{p \tau_4^2} - 1 \right) \\ &+ 2\eta_{2,0} \left(\frac{\tau_8}{p^2 \tau_4^2} - 1 \right) + 2\eta_{2,2} \left(\frac{\tau_6}{p^2 \tau_4^2} - 1 \right) \end{aligned}$$

TABLE II
DATASETS. $(K = 10^3, M = 10^6)$

Name	$\|L\|$	$\|R\|$	$\|E\|$	Type
Reuters	781K	283K	60.9M	Text
Gottron	556K	1.17M	83.6M	Misc
LiveJournal (LJ)	7.49M	3.20M	112M	Affiliation
Deli-ut	833K	4.51M	301M	Interaction

VI. EXPERIMENTS

In this section, we implement all proposed algorithms in C++ and compile with full optimization, where the code can be found at [2]. The implementation of competitors is obtained from their inventors. All experiments are conducted on a Linux machine. We test the proposed methods under the two-pass model, random-arrival model, and sliding-window model. We compare their performance under the different space settings.

A. Experimental Settings

Datasets. We use four real large bipartite networks, which are available in KONECT [1]. These bipartite networks are commonly used in previous work [16], [28], [50]. As shown in Table II, it includes *Reuters*, *LiveJournal*, *Gottron* and *Deli-ut*. In particular, the Reuters dataset is a bipartite network representing story–word inclusions from documents in the Reuters Corpus. The Gottron dataset consists of 556,000 text documents from TREC Disks 4 and 5, with a vocabulary of 1.1 million unique words. The LiveJournal dataset captures a bipartite network of users and their group memberships. The deli-ut dataset represents user–tag relationships sourced from http://delicious.com/. Before experimental evaluation, all duplicate edges in the graph are removed.

Metrics. Following the previous works [16], [29], we use the *Absolute Relative Error* (ARE) to measure the relative errors of estimation Y with respect to true value $\bowtie$. The metric ARE is defined as

$$ARE(Y) = |\frac{Y - \bowtie}{\bowtie}|.$$

Note that the smaller the score ARE, the higher the accuracy of the estimation.

For the sliding window model, we also use the *valid sample size* as a metric for algorithm performances, following the work [11]. The valid sample size is the number of edges of the sampled graph. With the same memory usage, the valid sample size indicates the capacity of algorithms to extract valid information by sampling.

Algorithms. For the two-pass model (resp. random arrival model), we evaluate our VS-2PASS (resp. VS-1PASS) against alternative algorithms CAS and sGrapp. For the sliding window model, we compare the performances of our B-TSW and VS-TSW methods with the algorithm FLEET-TSW [29]. Next, we provide a brief introduction to each alternative algorithm mentioned above.

- **CAS** [16]. The algorithm uses the number of co-affiliations between vertex pairs on one side of the graph to compute the estimation, where the co-affiliation is defined as the wedge on a particular side. For each arriving edge, CAS updates the total co-affiliations number and the co-affiliations number of related vertex pairs before sampling it with probability p. It samples a butterfly with probability p^2. We use the same setting for the AMS sketch of the CAS method and the proposed methods. CAS is the state-of-the-art unbiased algorithm with theoretical guarantees, proven to outperform FLEET 1-3 [29] according to the previous work [16].
- **sGrapp** [33]. It uses tumbling windows for estimation. Tumbling windows are a set of disjoint windows that cover the whole graph stream. In sGrapp, each window contains a fixed max number of unique timestamps. To compute estimation, sGrapp counts butterflies within a window and inter-window separately. It computes the exact butterfly count for each window and estimates the approximate count for inter-window butterflies as $|E|^{\alpha}$, where $|E|$ is the number of received edges and α is the approximation exponent parameter.
- **FLEET-TSW** [29]. It uses T reservoirs $R_1, \cdots R_T$. Each edge sampled into R_{i-1} is sampled into R_i with probability p. When a reservoir is full, the oldest edge in it will be removed. When queried for an estimation, it uses the first reservoir that has not deleted an edge of the current window to calculate the estimation.

B. Experimental results

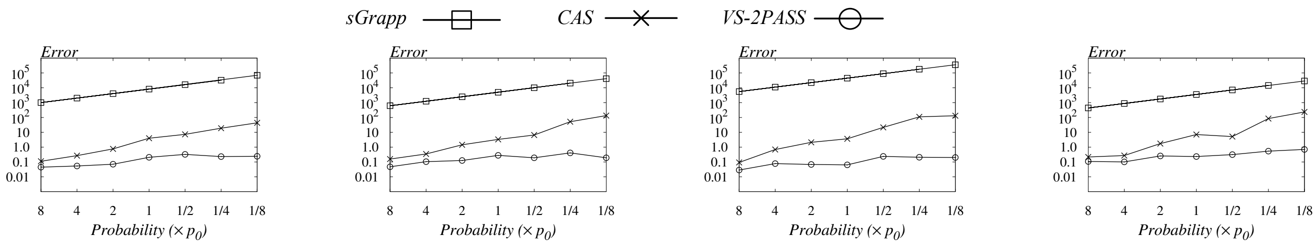
Two-pass model. In the first set of experiments, we compare the estimation accuracy of our VS-2PASS with its competitors CAS, sGrapp under the different memory usage settings on four bipartite datasets. The timestamp for each edge starts from 1 and increases by 1 for each received edge. As analyzed in Section III, the memory usage of VS-2PASS consists of the space used for storing sampled edges and AMS sketch. We denote the two parts of the memory as M_E and M_A, respectively. In the experiments, the number of edges within each time unit is the same; hence, the memory for each tumbling window M_{TW} of sGrapp is fixed. Following previous work [16], we make the memory usage for each algorithm the same by setting the sampling probability parameter p and setting the $M_E = M_A$ for CAS and VS-2PASS. We also vary the memory usage of sGrapp M_{TW} according to the space used by VS-2PASS to achieve a fair comparison. To obtain stable results, all the results are averaged over 100 runs for each algorithm and each setting. We use the median of the errors of the 10 tests to obtain the data points. The probability p_0 is set as 10^{-6} in the experiments.

We report the estimation performance ARE of each algorithm , with different sampling probability p in Figure 4. As shown in Figure 4, the estimation error of our VS-2PASS is much lower than other algorithms on all datasets. When p reaches $1/8 \times p_0$, the algorithm VS-2PASS achieves error rates 700x lower than the competitor CAS on the LJ dataset. We also note that the error rates of sGrapp are always the highest, as it relies on edge inputs needing to follow a specific distribution, making it inapplicable in many cases. The above experimental results indicate that our solution VS-2PASS outperforms other methods under the two-pass model when memory space is limited. It confirms that VS-2PASS increases the sampling probability of a butterfly compared to CAS.

Random arrival model. In the second set of experiments, we compare the estimation accuracy of our proposed VS-1PASS with the competitors CAS and sGrapp, under the random-arrival model. For fair comparison, We unified the space cost for each method, as described in Section VI-B. For each of four datasets, we shuffle the graph edges such that the graph streaming follow the definition of the random-arrival model. Then, the timestamp for each edge starts from 1 and increases by 1 for each received edge.

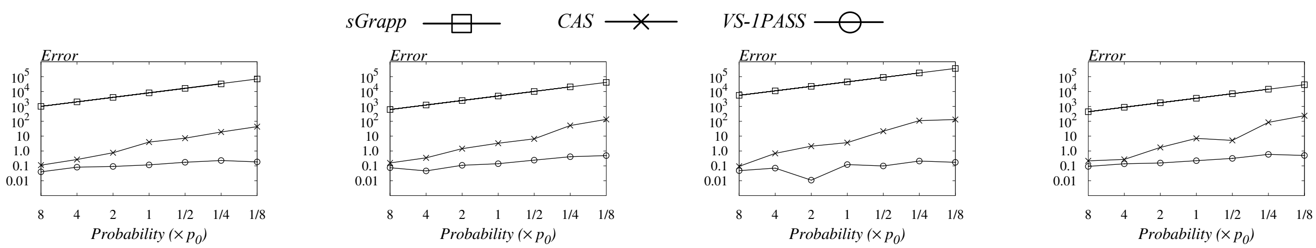
The experimental results are shown in Figure 5. From the figures, we can observe a similar trend as that of VS-2PASS. The error rate of our proposed method VS-1PASS is significantly lower than that of its competitors, i.e., CAS and sGrapp. This indicates that our proposed method VS-1PASS achieves better performance than the state-of-the-art algorithms under the random arrival model when the memory space is limited. It can be explained that the solution VS-1PASS has a higher butterfly sampling probability than the methods CAS and sGrapp.

Sliding Window Model. Then, we further test our proposed solutions, B-TSW and VS-TSW against the alternative FLEET-TSW, under the time-based sliding window model in terms of the estimation accuracy. We set the sample rate as $|E^{(s)}|/N$, i.e., the ratio of the expectation of sampled edge number to the window size N. We report the error rate ARE and the valid sample size as the performance metrics of each algorithm. All tested algorithms are set with the



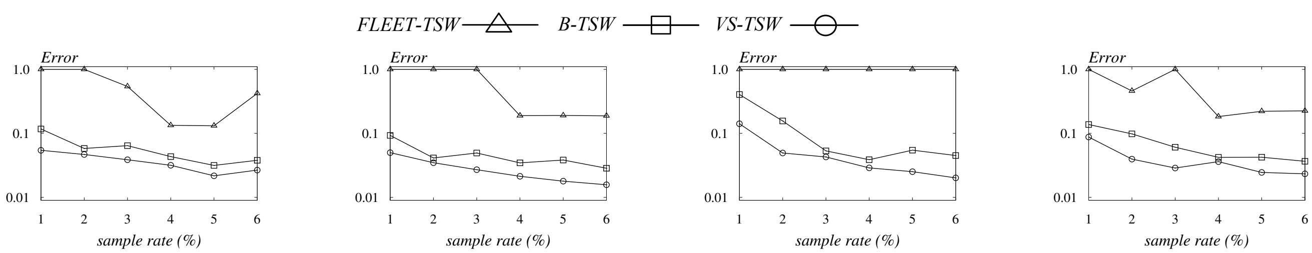
(a) Reuters (b) Gottron (c) LJ (d) Deli-ut

Fig. 4. Relative error of VS-2PASS in comparison with CAS and sGrapp at different sampling probabilities



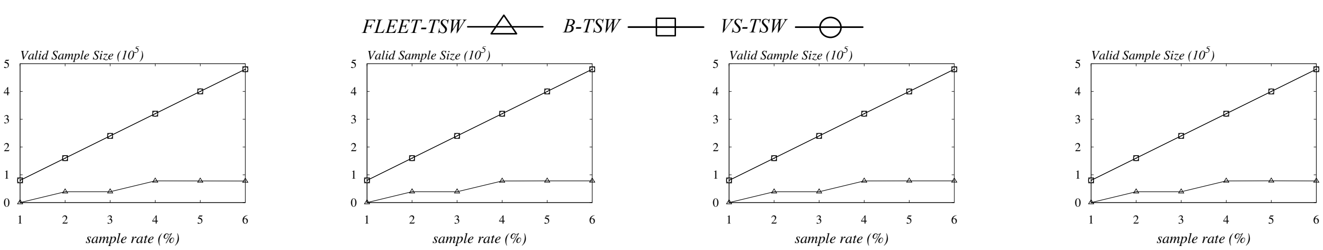
(a) Reuters (b) Gottron (c) LJ (d) Deli-ut

Fig. 5. Relative error of VS-1PASS in comparison with CAS and sGrapp at different sampling probabilities



(a) Reuters (b) Gottron (c) LJ (d) Deli-ut

Fig. 6. Relative error of B-TSW and VS-TSW in comparison with FLEET-TSW at different sample rates.

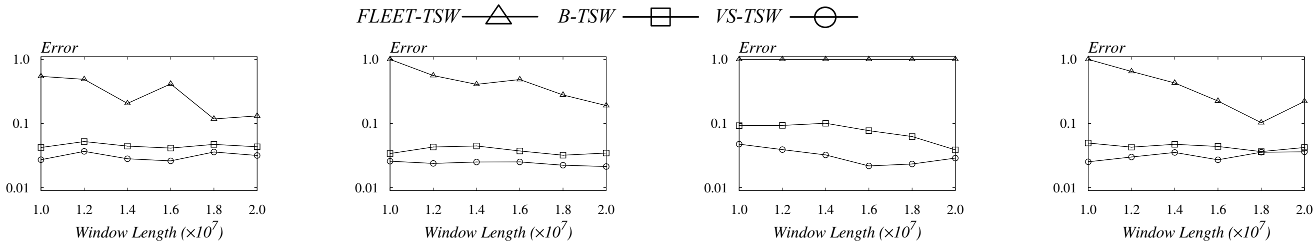


(a) Reuters (b) Gottron (c) LJ (d) Deli-ut

Fig. 7. Valid sample size of B-TSW and VS-TSW in comparison with FLEET-TSW at different sample rates

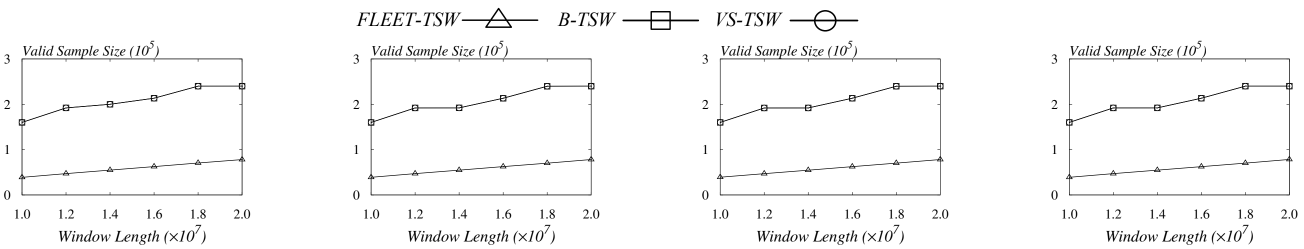
same memory usage. Recall that the memory usage of our algorithm is determined by the number of substreams, k. For each substream, at most two edges are stored. Therefore, we let $k = \frac{M_R}{2}$, where M_R is the space cost of FLEET-TSW. The vertex sampling probability of VS-TSW is set to 0.3. Time labels are attached to each edge, starting from 1 and getting increased by 1 with each edge received. We fix the window length at 2×10^7 and vary the sampling rate. The ARE and valid sample size for each data point are calculated as the average across all recorded windows.

The experimental results are shown in Figures 6 and 7. To obtain stable results, all the results are averaged over 5 runs for each algorithm. From Figure 6, we can observe that across all tested sampling rates, our proposed algorithms B-TSW and VS-TSW have superior performance in terms of ARE score, demonstrating their better capability in estimation accuracy. In addition, at lower sampling rates, VS-TSW has a significant accuracy advantage over B-TSW. This is consistent with the results of theoretical analysis. Then, from Figure 7, the proposed solutions B-TSW and VS-TSW can always



(a) Reuters (b) Gottron (c) LJ (d) Deli-ut

Fig. 8. Relative error of B-TSW and VS-TSW in comparison with FLEET-TSW at different window lengths



(a) Reuters (b) Gottron (c) LJ (d) Deli-ut

Fig. 9. Valid sample size of B-TSW and VS-TSW in comparison with FLEET-TSW at different window lengths

obtain a much higher number of valid samples in comparison with FLEET-TSW. These results demonstrate their favorable information extraction capability.

Further, we conduct experiments to examine the impact of the window size in the estimation accuracy performance of each algorithm. We fix the sample rate to 0.04 while varying the window length from 1×10^7 to 2×10^7, increasing by 2×10^6. The experimental results with different window sizes are shown in Figures 8 and 9. Again, from the figures, we can see that our proposed solution B-TSW and VS-TSW keep outperforming its competitor FLEET-TSW on all datasets. Furthermore, the improved version VS-TSW achieve better performance than its basic version, B-TSW. This shows that VS-TSW is the favorable choice..

VII. More Related Work

Motifs, also known as graphlets, are small cohesion units in a graph. Graphlet counting meets various needs in graph analysis and has numerous applications in practice. There has been considerable work [3], [6], [18], [23], [26], [44] on approximating triangle counting in graph streams. Triangle counting in the two or constant passes model has been extensively researched [5], [6], [23]. There are also works [8], [22] aimed at proposing approximate triangle counting algorithms for the random arrival model. Notably, the Doulion algorithm [38] estimates the number of triangles via a subgraph sampling strategy, sampling each edge with probability p, thus sampling each triangle with a probability of p^3. Rasmus et al. [25] introduced a colorful sampling of subgraph sampling, where the probability of sampling a triangle increases to p^2. Sanei-Mehri et al. [28] extended these methods to butterfly counting, obtaining methods that sample a butterfly with probabilities of p^4 and p^3, respectively. Our vertex sparsification technique can effectively increase this probability to p^2, with the same theoretical space cost.

Bipartite graphs can be regarded as a special case of general graphs. Motif counting and enumeration on bipartite graphs have recently attracted significant research attention. Various studies have explored distinct scenarios of butterfly counting, such as on GPU [48], data streaming [29], [33], uncertain graphs [54], among others. The work [42] studies how to maintain the number of butterflies in a batch-dynamic graph based on the previous work [40]. In [28] the authors also study how to adopt existing subgraph sampling algorithms for approximate triangle counting to approximate butterfly counting. This approach mainly limits the space consumption of the algorithm and is more suitable for streaming scenarios for approximate counting. In the literature, the approximate counting method [38] based on subgraph sampling was found to require sampling a large proportion of the original graph to produce reasonable estimates. The intuitive reason is that subgraph sampling is a more general technique. Wu et al. [47] point out that approximate counting methods based on subgraph sampling require sampling a large proportion of the original graph to obtain a reasonable estimate. Compared to subgraph sampling algorithms, local sampling algorithms find a wider range of real-world applications [28]. Hence, this paper focuses on the algorithm based on local sampling. In addition to butterfly and bi-triangle counting, the work [49] also studies (p, q)-biclique counting and enumeration. The work [53] investigates how to enumerate all bicliques efficiently. In [20], it studies how to enumerate maximum biclique. In the context of cohesive subgraph mining, research has been conducted on butterfly-based bi-truss decomposition [41], [43]. The paper [30] presents a hierarchical structure for modeling dense subgraphs based on the butterfly motif.

VIII. Conclusion

In this paper, we investigate the problem of approximate butterfly counting in various streaming scenarios. We propose several effective algorithms for approximate butterfly counting. We provide a rigorous theoretical analysis of the proposed methods. Extensive experiments have demonstrated the effectiveness of the proposed methods. In future work, we plan to explore the problem of approximate counting of other motifs in streaming scenarios.

References

[1] KONECT. http://konect.cc/networks/, 2013.

[2] Technical Report and Codes. https://github.com/qtguo/approx-butterfly-counting.git, 2024.

[3] N. K. Ahmed, N. G. Duffield, J. Neville, and R. R. Kompella. Graph sample and hold: a framework for big-graph analytics. In *KDD*, pages 1446–1455, 2014.

[4] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. In *STOC*, pages 20–29, 1996.

[5] S. Assadi, G. Kol, R. R. Saxena, and H. Yu. Multi-pass graph streaming lower bounds for cycle counting, max-cut, matching size, and other problems. In *FOCS*, pages 354–364, 2020.

[6] L. S. Buriol, G. Frahling, S. Leonardi, A. Marchetti-Spaccamela, and C. Sohler. Counting triangles in data streams. In *PODS*, pages 253–262, 2006.

[7] A. Chakrabarti, P. Ghosh, A. McGregor, and S. Vorotnikova. Vertex ordering problems in directed graph streams. In *SODA*, pages 1786–1802, 2020.

[8] G. Cormode and H. Jowhari. A second look at counting triangles in graph streams (corrected). *Theor. Comput. Sci.*, 683:22–30, 2017.

[9] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. Graph distances in the data-stream model. *SIAM J. Comput.*, 38(5):1709–1727, 2008.

[10] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. *Discrete mathematics & theoretical computer science*, 2007.

[11] X. Gou and L. Zou. Sliding window-based approximate triangle counting over streaming graphs with duplicate edges. In *SIGMOD*, pages 645–657, 2021.

[12] M. Kapralov, S. Khanna, and M. Sudan. Approximating matching size from random streams. In *SODA*, pages 734–751, 2014.

[13] C. Konrad, F. Magniez, and C. Mathieu. Maximum matching in semi-streaming with few passes. In *International Workshop on Approximation Algorithms for Combinatorial Optimization*, pages 231–242, 2012.

[14] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. RCV1: A new benchmark collection for text categorization research. *J. Mach. Learn. Res.*, 5:361–397, 2004.

[15] J. Li, D. Maier, K. Tufte, V. Papadimos, and P. A. Tucker. No pane, no gain: efficient evaluation of sliding-window aggregates over data streams. *SIGMOD Rec.*, 34(1):39–44, 2005.

[16] R. Li, P. Wang, P. Jia, X. Zhang, J. Zhao, J. Tao, Y. Yuan, and X. Guan. Approximately counting butterflies in large bipartite graph streams. *IEEE Trans. Knowl. Data Eng.*, 34(12):5621–5635, 2022.

[17] Z. Li, X. Shen, Y. Jiao, X. Pan, P. Zou, X. Meng, C. Yao, and J. Bu. Hierarchical bipartite graph neural networks: Towards large-scale e-commerce applications. In *ICDE*, pages 1677–1688, 2020.

[18] Y. Lim, M. Jung, and U. Kang. Memory-efficient and accurate sampling for counting local triangles in graph streams: From simple to multi-graphs. *ACM Trans. Knowl. Discov. Data*, 12(1):4:1–4:28, 2018.

[19] P. G. Lind, M. C. González, and H. J. Herrmann. Cycles and clustering in bipartite networks. *Physical review E*, 72(5):056127, 2005.

[20] B. Lyu, L. Qin, X. Lin, Y. Zhang, Z. Qian, and J. Zhou. Maximum biclique search at billion scale. *Proc. VLDB Endow.*, 13(9):1359–1372, 2020.

[21] G. S. Manku and R. Motwani. Approximate frequency counts over data streams. *Proc. VLDB Endow.*, 5(12):1699, 2012.

[22] A. McGregor and S. Vorotnikova. Triangle and four cycle counting in the data stream model. In *PODS*, pages 445–456, 2020.

[23] A. McGregor, S. Vorotnikova, and H. T. Vu. Better algorithms for counting triangles in data streams. In *PODS*, pages 401–411, 2016.

[24] M. D. Ornstein. Interlocking directorates in canada: evidence from replacement patterns. *Social Networks*, 4(1):3–25, 1982.

[25] R. Pagh and C. E. Tsourakakis. Colorful triangle counting and a mapreduce implementation. *Inf. Process. Lett.*, 112(7):277–281, 2012.

[26] A. Pavan, K. Tangwongsan, S. Tirthapura, and K. Wu. Counting and sampling triangles from a graph stream. *Proc. VLDB Endow.*, 6(14):1870–1881, 2013.

[27] G. Robins and M. Alexander. Small worlds among interlocking directors: Network structure and distance in bipartite graphs. *Comput. Math. Organ. Theory*, 10(1):69–94, 2004.

[28] S. Sanei-Mehri, A. E. Sariyüce, and S. Tirthapura. Butterfly counting in bipartite networks. In *SIGKDD*, pages 2150–2159, 2018.

[29] S. Sanei-Mehri, Y. Zhang, A. E. Sariyüce, and S. Tirthapura. FLEET: butterfly estimation from a bipartite graph stream. In *CIKM*, pages 1201–1210, 2019.

[30] A. E. Sariyüce and A. Pinar. Peeling bipartite networks for dense subgraph discovery. In *WSDM*, pages 504–512, 2018.

[31] R. R. Saxena, N. Singer, M. Sudan, and S. Velusamy. Streaming complexity of csps with randomly ordered constraints. In *SODA*, pages 4083–4103, 2023.

[32] T. Schank and D. Wagner. Approximating clustering coefficient and transitivity. *Journal of Graph Algorithms and Applications*, 9(2):265–275, 2005.

[33] A. Sheshbolouki and M. T. Özsu. sgrapp: Butterfly approximation in streaming graphs. *ACM Trans. Knowl. Discov. Data*, 16(4):76:1–76:43, 2022.

[34] K. Shin, S. Oh, J. Kim, B. Hooi, and C. Faloutsos. Fast, accurate and provable triangle counting in fully dynamic graph streams. *ACM Trans. Knowl. Discov. Data*, 14(2):12:1–12:39, 2020.

[35] J. Sun, H. Qu, D. Chakrabarti, and C. Faloutsos. Neighborhood formation and anomaly detection in bipartite graphs. In *ICDM*, pages 418–425, 2005.

[36] A. Tanay, R. Sharan, and R. Shamir. Discovering statistically significant biclusters in gene expression data. *Bioinformatics*, 18(suppl_1):S136–S144, 2002.

[37] Y. Tong, J. She, B. Ding, L. Chen, T. Wo, and K. Xu. Online minimum matching in real-time spatial data: Experiments and analysis. *Proc. VLDB Endow.*, 9(12):1053–1064, 2016.

[38] C. E. Tsourakakis, U. Kang, G. L. Miller, and C. Faloutsos. DOULION: counting triangles in massive graphs with a coin. In *SIGKDD*, pages 837–846, 2009.

[39] J. Wang, A. W. Fu, and J. Cheng. Rectangle counting in large bipartite graphs. In *IEEE International Congress on Big Data*, pages 17–24, 2014.

[40] K. Wang, X. Lin, L. Qin, W. Zhang, and Y. Zhang. Vertex priority based butterfly counting for large-scale bipartite networks. *Proc. VLDB Endow.*, pages 1139–1152, 2019.

[41] K. Wang, X. Lin, L. Qin, W. Zhang, and Y. Zhang. Efficient bitruss decomposition for large-scale bipartite graphs. In *ICDE*, pages 661–672, 2020.

[42] K. Wang, X. Lin, L. Qin, W. Zhang, and Y. Zhang. Accelerated butterfly counting with vertex priority on bipartite graphs. *VLDB J.*, 2022.

[43] K. Wang, X. Lin, L. Qin, W. Zhang, and Y. Zhang. Towards efficient solutions of bitruss decomposition for large-scale bipartite graphs. *VLDB J.*, 31(2):203–226, 2022.

[44] P. Wang, Y. Qi, Y. Sun, X. Zhang, J. Tao, and X. Guan. Approximately counting triangles in large graph streams including edge duplicates with a fixed memory usage. *Proc. VLDB Endow.*, 11(2):162–175, 2017.

[45] Y. Wang, Q. Fan, Y. Li, and K. Tan. Real-time influence maximization on dynamic social streams. *Proc. VLDB Endow.*, 10(7):805–816, 2017.

[46] D. J. Watts and S. H. Strogatz. Collective dynamics of ‘small-world’networks. *nature*, 393(6684):440–442, 1998.

[47] B. Wu, K. Yi, and Z. Li. Counting triangles in large graphs by random sampling. *IEEE Trans. Knowl. Data Eng.*, 28(8):2013–2026, 2016.

[48] Q. Xu, F. Zhang, Z. Yao, L. Lu, X. Du, D. Deng, and B. He. Efficient load-balanced butterfly counting on GPU. *Proc. VLDB Endow.*, 15(11):2450–2462, 2022.

[49] J. Yang, Y. Peng, and W. Zhang. (p, q)-biclique counting and enumeration for large sparse bipartite graphs. *Proc. VLDB Endow.*, 15(2):141–153, 2021.

[50] Y. Yang, Y. Fang, M. E. Orlowska, W. Zhang, and X. Lin. Efficient bi-triangle counting for large bipartite networks. *Proc. VLDB Endow.*, pages 984–996, 2021.

[51] F. Zhang, D. Chen, S. Wang, Y. Yang, and J. Gan. Scalable approximate butterfly and bi-triangle counting for large bipartite networks. *Proc. ACM Manag. Data*, 1(4):259:1–259:26, 2023.

[52] G. Zhang, Z. Li, J. Huang, J. Wu, C. Zhou, J. Yang, and J. Gao. efraudcom: An e-commerce fraud detection system via competitive graph neural networks. *ACM Trans. Inf. Syst.*, 40(3):47:1–47:29, 2022.

[53] Y. Zhang, C. A. Phillips, G. L. Rogers, E. J. Baker, E. J. Chesler, and M. A. Langston. On finding bicliques in bipartite graphs: a novel algorithm and its application to the integration of diverse biological data types. *BMC bioinformatics*, 15(1):1–18, 2014.

[54] A. Zhou, Y. Wang, and L. Chen. Butterfly counting on uncertain bipartite networks. *Proc. VLDB Endow.*, 15(2):211–223, 2021.