FLEET: Butterfly Estimation from a Bipartite Graph Stream

Seyed-Vahid Sanei-Mehri
Iowa State University
vas@iastate.edu

Yu Zhang
Iowa State University
yuz1988@iastate.edu

Ahmet Erdem Sarıyüce
University at Buffalo
erdem@buffalo.edu

Srikanta Tirthapura
Iowa State University
snt@iastate.edu

ABSTRACT

We consider space-efficient single-pass estimation of the number of butterflies, a fundamental bipartite graph motif, from a massive bipartite graph stream where each edge represents a connection between entities in two different partitions. We present a space lower bound for any streaming algorithm that can estimate the number of butterflies accurately, as well as FLEET, a suite of algorithms for accurately estimating the number of butterflies in the graph stream. Estimates returned by the algorithms come with provable guarantees on the approximation error, and experiments show good tradeoffs between the space used and the accuracy of approximation. We also present space-efficient algorithms for estimating the number of butterflies within a sliding window of the most recent elements in the stream. While there is a significant body of work on counting subgraphs such as triangles in a unipartite graph stream, our work seems to be one of the few to tackle the case of bipartite graph streams.

KEYWORDS

butterfly counting, rectangle counting, data stream

ACM Reference Format:
Seyed-Vahid Sanei-Mehri, Yu Zhang, Ahmet Erdem Sarıyüce, and Srikanta Tirthapura. 2019. FLEET: Butterfly Estimation from a Bipartite Graph Stream. In *The 28th ACM International Conference on Information and Knowledge Management (CIKM '19), November 3–7, 2019, Beijing, China.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3357384.3357983

1 INTRODUCTION

Enumeration and counting of graph substructures has emerged as a basic tool in understanding complex networks, and has found wide applications in social networks, spam/fraud detection, and link recommendation, and more. Due to the scale of today's datasets, enumeration and counting needs to be performed on very large graphs, with the order of billions of vertices and trillions or larger number of graph substructures. Such large graphs are naturally modeled as graph streams – the edges of such a graph are not available all at once, but are instead observed as a sequence of updates.

In this work, we focus on **bipartite graph streams**. A bipartite graph consists of two disjoint node sets L and R. Each edge in the graph connects a node in L with a node in R. Bipartite graphs are widely used in modeling relationships in the real world. For instance, they can be used to model relationships between authors and papers they have published, where the set of authors form one node partition, papers form the other node partition, and an author has an edge to each paper that she published [17]. In web search, bipartite graphs have been used in modeling relations between queries and URLs in query logs [27] and in matching users to advertisements in computational advertising [2, 32]. In computational biology, bipartite graphs are used to model enzyme-reaction links in metabolic pathways and gene-disease associations [37]. Other examples include user-product relations, word-document affiliations, and actor-movie networks. Bipartite graphs can be used to represent hypergraphs that capture many-to-many relations among entities, through having the hyperedges in one partition, and the entities in another partition. Bipartite graph streams are natural in the above examples, where new entities may arise in either partition, and new edges are observed as time progresses. The challenge in bipartite graph stream processing is to maintain properties in a time- and space-efficient way as more edges are observed.

While there is a rich literature on subgraph motif counting from unipartite network streams, these methods do not take into account the special structure present in bipartite networks. For instance, the number of triangles (cliques of size 3), a widely studied metric for unipartite graph streams [3, 6, 8, 13, 15, 21–23, 25, 28, 36, 45, 47–49, 53], is not a useful metric for bipartite networks, since a bipartite network is triangle-free. Instead, the most basic motif which models cohesion in a bipartite network is the 2×2 biclique, known as a butterfly [5, 41, 42] or a rectangle [50]. The number of butterflies has been used in defining the clustering coefficient in a bipartite graph [29, 40] and can be considered as playing the same role in bipartite networks as the triangle did in unipartite networks – a building block for community structure. Though there are some prior works on counting butterflies in a static bipartite graph [41, 50], these have not considered bipartite graph streams.

1.1 Contributions

We present FLEET, butterFLy Estimation from a bipartitE graph sTream – a suite of space-efficient one-pass streaming algorithms for estimating the number of butterflies in a bipartite graph stream. Our algorithms use fixed-size memory that is much smaller than the size of the stream, and continuously maintain an estimate of the

CIKM '19, November 3–7, 2019, Beijing, China
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-6976-3/19/11…$15.00
https://doi.org/10.1145/3357384.3357983

number of butterflies as edges arrive in the stream. Our algorithms are simple to implement, backed up by theoretical guarantees, and have good practical performance.

– **Space Lower Bound.** We first show a lower bound, proving that any streaming algorithm (whether deterministic or randomized) that can approximately maintain an estimate of the number of butterflies with a bounded relative error in a graph on n vertices must use a memory of size $\Omega(n^2)$ on certain input streams. Note that using $\Omega(n^2)$ memory, it is possible to store the entire graph stream. This shows that in general, it is not possible for a streaming algorithm to maintain an estimate of the number of butterflies using memory sub-linear in the size of the graph. However, the lower bound applies for cases when the number of butterflies in the graph is very small; in particular, the proof depends on distinguishing between two cases, one where there are no butterflies, and another where there is a single butterfly. Real-world bipartite graph streams typically have a large number of butterflies (e.g., Figure 3), and hence, one cannot rule out algorithms that are more space-efficient and return an accurate estimate of the number of butterflies, thus motivating our further work on small-memory algorithms.

– **Infinite Window Streaming.** We next present small-memory algorithms for estimating the number of butterflies within an *infinite window* consisting of all edges seen so far in the stream. The memory used by our algorithms is no more than a given parameter M. We first present an algorithm Fleet1 that is based on adaptive random sampling from the edge stream so that as more edges are seen, the sampling probability decreases so as to fit within available memory. We prove that the estimator returned by Fleet1 is unbiased, and derive concentration bounds showing that the estimator is close to the actual value with a high probability, if the memory used is large enough. We present two enhancements to Fleet1, leading to algorithms Fleet2 and Fleet3 which provide better memory-accuracy tradeoffs in practice.

– **Sliding Window Streaming.** In stream data mining, the scope of aggregation often needs to be restricted to include edges that have arrived within a recent window. To handle such cases, we present extensions of Fleet to the sliding window model [7, 12, 19]. We consider two types of sliding windows. (1) For a *sequence-based* window, defined as the set of W most recent edges in the stream for a window size parameter W, we present an algorithm FleetSSW (2) For a *time-based* window, defined as stream elements whose timestamps are greater than $(c - w)$, where c is the current time and w is the window size, we present an algorithm FleetTSW. Both algorithms use a bounded memory that does not increase with the number of edges in the window. Our algorithm for a time-based window is flexible to receive the window size as a parameter during the query, and does not need to know the window size in advance.

– **Experimental Evaluation.** We experimentally evaluate Fleet on real-world graph streams. Results show that our algorithms are effectively able to handle large graph streams. For instance, on the Bag-pubmed graph with approximately 500M edges and 40T butterflies, our algorithms are able to achieve estimates with an error of less than 1% using a memory of 600K edges. Our methods present different tradeoffs between memory, estimation accuracy, and runtime, that make them applicable in real-world applications with different requirements, and significantly outperform prior works on subgraph counting from graph streams [3, 9, 31].

1.2 Related Works

Network motifs. Network motifs are small subgraphs that are defined on a few nodes and edges. Unlike graph communities or dense subgraphs, whose sizes do not have to be bounded, network motifs are typically subgraphs with less than six nodes. A similar concept is graphlets [39]. Network motif detection and counting is now an indispensable tool in network analysis [30, 34]. The distribution of motif counts in a network, as well as the number of motifs that a node takes part in, help characterize the roles of networks and nodes [33], an idea that has been used in numerous applications in networking, web and social network analysis, and computational biology [8, 18, 43, 46, 50].

Butterfly Counting. There have been relatively few works on counting motifs in a bipartite graph. Wang et al. [50] presented exact algorithms for butterfly counting in static graphs that outperform generic matrix multiplication based methods [6]. Sanei-Mehri et al. [41] and Zhu et al. [54] present exact and randomized algorithms for butterfly counting on static bipartite graphs. [44, 51] presents parallel algorithms for butterfly counting on static graphs. All these works have considered static graphs and not graph streams, like we do here.

Motif Counting in Graphs. There are a number of algorithms known for triangle counting for unipartite streaming graphs, including [3, 8, 13, 15, 21–23, 28, 36, 45, 52, 53]. Recent works on counting 4-vertex [4] and 5-vertex subgraphs [38] have focused on exact counting and are not designed for streaming graphs.

Since a butterfly is a 4-cycle, prior works on counting 4-cycles in *any* graph stream [9, 10, 14, 24, 31] can be also applied to a bipartite graph stream, and we compare with such prior work. Note that these algorithms do not use the additional structure in a bipartite graph (absence of edges between vertices in the same partition), and are thus naturally disadvantageous for bipartite streams. Bordino et al. [10] present three-pass algorithms for counting 4-cycles, while we focus on single-pass streaming algorithms. Buriol et al. [14] consider 3,3-biclique counting from a stream. They assume the *incidence stream* model, where edges in the graph stream are presented in a specific order such that all edges incident to a given vertex arrive together, whereas we assume the more general model where edges can arrive in an arbitrary order. Bera and Chakrabarti [9] present algorithms for counting 4-cycles in a graph using two passes through the stream. The work of Ahmed et al. [3] can be specialized to count butterflies in a stream, and we present an experimental comparison with [3, 9] in Section 6. Other works include Manjunath et al. [31] and Kane et al. [24] for subgraph counting based on graph sketches.

Lower Bounds for Subgraph Counting. There are multiple prior works on memory (space) lower bounds for triangle and subgraph counting in general graphs [9, 11, 16, 53], but not for subgraph counting in bipartite graphs. Since a bipartite graph is more restrictive than a general graph (certain edges are disallowed), lower bounds for unipartite graphs do not directly apply to bipartite graphs. To the best of our knowledge, our work presents the first lower bounds for subgraph counting in bipartite graph streams.

2 PRELIMINARIES

We consider simple unweighted and undirected bipartite graphs, without multiple edges between the same pair of vertices. Let $G =$

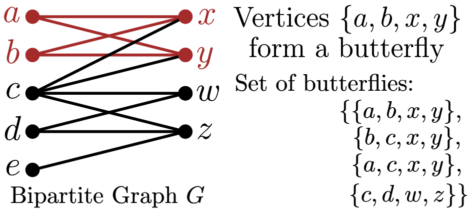


Figure 1: Butterflies in a bipartite graph. Graph G contains four butterflies.

(V, E) be a bipartite graph with vertices V and edges E. The vertex set V of G is partitioned into two disjoint sets L and R. The edge set $E \subseteq L \times R$, so each edge e connects one vertex in set L and the other in set R. A **butterfly** is subgraph on four vertices $\{a, b, x, y\} \subset V$, where $a, b \in L$ and $x, y \in R$ such that edges $(a, x), (a, y), (b, x)$ and (b, y) exist in the edge set E (see Figure 1).

A **graph stream** is a sequence of edges $\mathcal{S} = e_1, e_2, \ldots$ where e_i is the i-th edge in the stream. For $t > 0$, let $\mathcal{S}^t$ denote the first t edges of the stream and let $G^t = (V^t, E^t)$ denote the graph formed by the first t edges, i.e., $G^t = \{e_1, e_2, \ldots, e_t\}$. Let $\bowtie^t(G)$ denote the set of all butterflies in the graph G^t and let $\xi^t(G) = |\bowtie^t(G)|$ denote the number of butterflies in G^t. When G is clear from the context, we use notations $\bowtie^t, \xi^t$, and when both G and t are clear from the context, we use $\bowtie, \xi$. Figure 2 shows the setup for processing a stream of edges from a bipartite network. We consider the following settings.

Infinite Window: For any $t > 0$ and a stream $\mathcal{S}$, the goal is to continuously maintain (an estimate of) ξ^t, the number of butterflies in the graph G^t, as t changes.

Sliding Window: For a window size parameter W, a *sequence-based sliding window* is defined as the set of W most recent edges. For $t \geq W$, when edge e_t is observed, the sliding window consists of edges $e_{t-W+1}, e_{t-W+2}, \ldots e_t$. For $t < W$, the window consists of the entire stream so far. The goal is to continuously maintain an estimate of the number of butterflies in the graph defined by the sliding window. We also consider *time-based sliding window*, a generalization of sequence-based window, defined as the set of edges whose timestamps are the most recent. In a time-based window, the window size does not correspond to a specific number of edges, but instead to a range of timestamps.

Our randomized algorithms solely rely on randomness internal to the algorithm, and do not assume that the input is drawn from a specific probability distribution. The input graph stream, including the set of edges and their order of arrivals, could be generated by an adversary. In our space complexity analysis, we assume that a single edge from the graph and an edge timestamp can be stored in a constant number of words. Let $[n]$ denote the set $\{1, 2, 3, \ldots, n\}$.

3 SPACE LOWER BOUND

We show that it is impossible for any streaming algorithm to approximate the number of butterflies to within a small relative error using $o(n^2)$ space. This shows that one cannot expect an algorithm that always returns an accurate estimate using fixed space. In fact, the lower bound shows that essentially, the entire graph needs to be stored (which is possible in $\Theta(n^2)$ bits), if one desires an algorithm that always returns an accurate estimate of the number of butterflies in a bipartite graph stream. Note that our lower bound applies to randomized as well as deterministic algorithms, and for algorithms that return either exact or approximate answers. Note that this lower bound is based on the space complexity of distinguishing between an edge stream that has zero butterflies and one that has at least butterfly.

Theorem 3.1. For any streaming algorithm $\mathcal{L}$ that estimates $\xi(G)$ for a streaming bipartite graph G on n vertices, there exist input graph streams on which the algorithm uses memory $\Omega(n^2)$ bits.

The proof uses a reduction from a one-round communication complexity problem, where there are two parties Alice and Bob. Alice gets input $a \in A$ and Bob gets input $b \in B$. It is required to compute a function $g(a, b)$ using only one-way communication from Alice to Bob, while communicating as few bits as possible. g may be computed approximately, and there may be a failure probability that the approximation error is not achieved. Let the one-round communication complexity of function $g : A \times B \to Z$ with failure probability δ be denoted by $R^1_\delta(g)$.

Consider function f that takes as input n sets $S_1, S_2, \ldots, S_n$ and two integers i and j where each $S_k, 1 \leq k \leq n$ is a subset of $[n]$ of size $n/10$ and $i, j \in [n]$. The function is defined as:

$$f(S_1, S_2, \ldots, S_n, i, j) = \begin{cases} 1 & \text{if} \quad j \in S_i \\ 0 & \text{otherwise} \end{cases}$$

Inputs $S_1, S_2, \ldots, S_n$ are given to Alice and inputs i and j are given to Bob. Communication is allowed only from Alice to Bob, and Bob has to return the approximate value of the function. We use the following result from [53].

Lemma 1 (Bar-Yossef et al. [53]). The one round communication complexity of any algorithm for f is lower bounded as follows: for any $0 < \delta < 1/100$, $R^1_\delta(f) \geq n^2/40$.

Proof of Theorem 3.1. Suppose there exists a streaming algorithm $\mathcal{L}$ for estimating the number of butterflies with relative error of $1/2$ with error probability no more than δ, which uses space of s bits. We reduce the one-round distributed computation of f to streaming butterfly counting as follows.

Given her input $S_1, S_2, \ldots, S_n$, Alice constructs a part of graph G on $4n$ vertices with vertex set $P \cup Q \cup R \cup T$, where $P = \{p_1, p_2, \ldots, p_n\}$, $Q = \{q_1, q_2, \ldots, q_n\}$ and similarly R and T. She inserts the following edges into the streaming algorithm in any order. First for each $k = 1 \ldots n$, edges (p_k, q_k) and (q_k, r_k) are inserted. Next, for each set $S_k, k = 1 \ldots n$, and for each $\ell \in S_k$, an edge is inserted between t_k and r_ℓ. After Alice is done inserting all these edges into $\mathcal{L}$, she transmits the contents of the memory of $\mathcal{L}$ (the entire current state) to Bob, incurring a communication cost of no more than s bits.

Upon receiving the state of $\mathcal{L}$ from Alice, Bob continues running $\mathcal{L}$ by inserting edge (t_i, p_j) into the graph G. He then queries $\mathcal{L}$ for an approximate count of the number of butterflies in G. If the answer is non-zero, then he declares that $f(S_1, S_2, \ldots, S_n, i, j) = 1$, and if the answer is zero, then he declares the function to be 0. Note that if $j \in S_i$, then there is an edge (t_i, r_j) inserted by Alice. If we further consider edges (t_i, p_j) inserted by Bob, and edges (p_j, q_j) and (q_j, r_j) inserted by Alice, we get a single butterfly in G. If $j \notin S_i$ then, it can be verified that there are no butterflies in G. Since $\mathcal{L}$ provides a relative error guarantee, it must return a non-zero estimate if the actual number of butterflies is non-zero and an estimate of zero if the actual number of butterflies is zero.

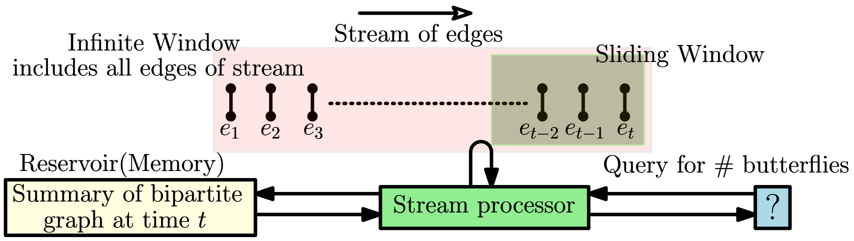


Figure 2: Set up for processing a graph stream.

We further note that G is a bipartite graph whose vertex set consists of two partitions, $P \cup R$ and $Q \cup T$. It can be verified that there are no edges connecting vertices within a single partition.

Thus, we have reduced the one-round distributed computation of f using s bits of communication to streaming butterfly counting using memory of s bits. Using Lemma 1, we see that $s \geq n^2/40$, thus completing the lower bound proof. □

4 INFINITE WINDOW STREAMING

We present streaming algorithms for estimating the number of butterflies over an infinite window, i.e., all edges seen so far. Our randomized algorithms maintain an unbiased estimate of the number of butterflies using a bounded memory of M, and provide a trade-off between memory used and the accuracy of the estimate.

4.1 Adaptive Sampling: FLEET1

We use random sampling from the stream of edges. An initial attempt uses Bernoulli Sampling (BERN) with parameter p, $0 < p \leq 1$. Each arriving edge is sampled into a reservoir with probability p. The number of butterflies among the sampled edges is incrementally maintained, and is multiplied by the appropriate normalization factor, to estimate the number of butterflies in the stream. The disadvantage of BERN is that it requires setting parameter p to the "right value", which depends on the input itself. If p is too small, the error in estimation can be large, and if p is too large, then the reservoir size can be very large.

Our first algorithm, FLEET1, solves this problem by adaptively setting p throughout the computation, so as to keep the memory bounded by M. Initially, p is set to 1, and all edges are sampled into the reservoir. When the size of the reservoir exceeds M, FLEET1 sub-samples by retaining each edge in the current reservoir with a probability γ. Further edges are sampled with probability γ. When the size of the reservoir again exceeds M, the same process is repeated, and further edges are sampled with probability γ^2, and so on. The size of the reservoir never exceeds M edges, and is at least γM, with high probability, except during the initial stages of the stream. FLEET1 also continuously maintains the number of butterflies among sampled edges. When an estimate is desired, the number of butterflies among the sampled edges is returned, after multiplying by the appropriate normalization factor.

Algorithm 1: FLEET1 ($\mathcal{S}, M$): Adaptive sampling

```
Input: Edge stream $\mathcal{S}$, max. reservoir size $M$, resampling
       parameter $\gamma$ (default value of $\gamma = 0.5$)
Output: Estimate of $\xi^t$, the number of butterflies at $t$
1  $p \leftarrow 1$, $\mathcal{R} \leftarrow \emptyset$, $t \leftarrow 0$, $\beta \leftarrow 0$
2  for each edge $e$ in $\mathcal{S}$ do
3      $t \leftarrow t + 1$
4      while $|\mathcal{R}| \geq M$ do
5          $p \leftarrow \gamma p$
6          for each edge $e \in \mathcal{R}$ do
7              Keep $e$ in $\mathcal{R}$ with prob. $\gamma$ and discard with prob. $1 - \gamma$
8          $\beta \leftarrow p^{-4} \times \xi(\mathcal{R})$ // number of butterflies in $\mathcal{R}$
9      if coin ($p$) is Head then
10         $\mathcal{R} \leftarrow \mathcal{R} \cup \{e\}$
11         $\beta \leftarrow \beta + p^{-4} \times$ BFC-EDGE($e$, $\mathcal{R}$)
12     $Y^t_{\text{FLEET1}} \leftarrow \beta$
```

Details are presented in Algorithm 1. Each time the reservoir is sub-sampled, FLEET1 uses an (exact) algorithm to compute $\xi(\mathcal{R})$, the number of butterflies in the reservoir using prior methods designed for a static graph, such as[41, 54]. FLEET1 also uses an algorithm BFC-EDGE(e, E) to count the number of butterflies that contain edge e in the graph induced by edge set E. This can be achieved using prior work such as [41]. For the purposes of the current discussion, the reader can assume that parameter γ is set to $1/2$ – the main advantages of our algorithm, including bounded sample size and provable accuracy still hold. A modest tradeoff between accuracy and runtime can be obtained by setting γ to other values between $1/2$ and 1, as we discuss further in Section 6.3. Lemma 2 shows that FLEET1 maintains an unbiased estimate of the butterfly count after observing each edge. Let Y^t_{FLEET1} denote the estimate of ξ^t returned by Algorithm 1 (Line 12) at time t.

Lemma 2. $E[Y^t_{\text{FLEET1}}] = \xi^t$

Proof. Suppose the butterflies in $\Join^t$ are numbered from 1 to ξ^t. Let $X^t_i (1 \leq i \leq \xi^t)$ be a random variable equal to 1 if all edges of the i^{th} butterfly appear in $\mathcal{R}$, and 0 otherwise. Let $X^t = \sum_{i=1}^{\xi^t} X^t_i$. From Line 11 of Algorithm 1, we have $Y^t_{\text{FLEET1}} = X^t \left(p^t\right)^{-4}$. When $t \leq M$, all edges of the stream are sampled, and $X^t = \xi^t$. When $t > M$, each edge appears in $\mathcal{R}$ with probability p^t. Note that p^t itself is a random variable equal to the sampling probability of an incoming edge at time t. To compute $\mathbb{E}[Y^t_{\text{FLEET1}}]$, we first condition on p^t and then remove the conditioning.

Since there are four edges in a butterfly, $\mathbb{E}\left[X^t_i \,\middle|\, p^t\right] = \left(p^t\right)^4$. With linearity of expectation, $\mathbb{E}\left[X^t \,\middle|\, p^t\right] = \sum_{i=1}^{\xi^t} \mathbb{E}\left[X^t_i \,\middle|\, p^t\right] = \xi^t \left(p^t\right)^4$. Then, $\mathbb{E}\left[Y^t_{\text{FLEET1}} \,\middle|\, p^t\right] = \mathbb{E}\left[X^t \left(p^t\right)^{-4} \,\middle|\, p^t\right] = \xi^t$. By conditional expectation, $\mathbb{E}\left[Y^t_{\text{FLEET1}}\right] = \mathbb{E}\left[\mathbb{E}\left[Y^t_{\text{FLEET1}} \,\middle|\, p^t\right]\right] = \xi^t$. □

Concentration analysis of Y^t_{FLEET1}: We next show that if the upper bound on the reservoir size (M) is large enough, then the estimate Y^t_{FLEET1} will be concentrated around its expectation, i.e., have a small relative error, with a high probability. There are a few complexities to deal with here. First, the sampling probability p^t

is itself a random variable, since it is the result a random process. We first analyze a simpler algorithm Bern(p) that is based on fixed sampling probability p, which we have explained earlier. After t edges, let Z_p^t denote the estimate computed by Bern(p), of $\xi(G^t)$, the number of butterflies in G^t. Lemma 3 shows a concentration bound for Z_p^t. The difficulty with analyzing Z_p^t is in handling the dependency between random variables corresponding to different butterflies being sampled into the reservoir. Though different edges are sampled independently by Bern(p), different butterflies are not necessarily independent since butterflies may share edges. We address this with the help of the Hajnal-Szemerédi theorem, building on ideas from prior works [25, 35, 47] all of who applied this idea in the context of triangle counting.

Lemma 3. After t edges, let h^t denote the maximum number of butterflies in G^t that an edge can be part of. For a fixed p, and for any $\epsilon \in (0, 1)$, we have $\Pr\left[\left|Z_p^t - \xi^t\right| \geq \epsilon\xi^t\right] \leq 8h^t \cdot exp\left(-\frac{\epsilon^2 \xi^t p^4}{12h^t}\right)$

Proof. Consider a new graph $H^t = (V_{H^t}, E_{H^t})$ constructed from G^t: each vertex $v \in V_{H^t}$ corresponds to a butterfly in G^t. For each pair of butterflies $u, v \in V_{H^t}$ that share at least one edge, there is an edge in E_{H^t}. Since each edge in G^t can be shared by at most h^t butterflies, the maximum degree of vertex $u \in V_{H^t}$ is $(4h^t - 4)$. Using the Hajnal-Szemerédi theorem [20], there exists an equitable coloring of H^t with at most $(4h^t - 3)$ colors. Let the colors be numbered as $\{1, 2, \ldots, 4h^t - 3\}$. For each butterfly $i \in V_{H^t}$, let random variable X_i be defined as: $X_i = 1$ if all edges of butterfly i are sampled by Bern at time t, and 0 otherwise. Let the set of butterflies assigned color j be denoted by C_j. For two butterflies $a, b \in C_j$, note that X_a and X_b are independent, since a and b do not share edges, and all their edges are sampled independently. Define $Y_j = \sum_{i \in C_j} X_i$. Note that Y_j is a Binomial random variable since it is the sum of independent 0-1 random variables. We have $\mathbb{E}[Y_j] = |C_j| \cdot p^4$ and $|C_j| > \xi^t/4h^t$, since the coloring of vertices in V_{H^t} is an equitable coloring. Using the Chernoff bound, $\Pr\left[\left|Y_j - |C_j| \cdot p^4\right| \geq \epsilon \cdot |C_j| \cdot p^4\right] \leq 2\exp\left(-\frac{\epsilon^2}{3} \cdot \frac{\xi^t}{4h^t} \cdot p^4\right)$

Using the union bound,
$\Pr\left[|Z_p^t - \xi^t| \geq \epsilon\xi^t\right] = \Pr\left[\left|\sum_{j=1}^{4h^t-3} Y_j \cdot p^{-4} - \sum_{j=1}^{4h^t-3} |C_j|\right| \geq \epsilon \sum_{j=1}^{4h^t-3} |C_j|\right]$
$\leq \sum_{j=1}^{4h^t-3} \Pr\left[||Y_j \cdot p^{-4} - |C_j|| \geq \epsilon|C_j|)\right] \leq 8h^t \cdot exp\left(-\frac{\epsilon^2 \xi^t p^4}{12h^t}\right)$ □

We next derive a concentration result for Fleet1. Fleet1 essentially returns the estimate due to Bern(p^t) where p^t is itself a random variable. While it is possible to compute the expected value of p^t, this cannot be directly plugged into the Lemma 3. Lemma 4 shows that for a large enough reservoir size, Fleet1 returns an estimate that is concentrated around its expectation. The proof considers multiple Bern(p) instances, one for each sampling probability, and combines with the event of Fleet1 stopping at one of these levels.

Lemma 4. Assume $\gamma \leq 0.9$. For any $\epsilon, \delta \in (0, 1)$, when M satisfies $M \geq 6t \cdot \sqrt[4]{\frac{12h^t}{\epsilon^2 \xi^t}} \cdot \ln \frac{8h^t(4+\delta)}{\delta}$ then $\Pr\left[\left|Y^t_{\text{Fleet1}} - \xi^t\right| \geq \epsilon\xi^t\right] \leq \delta$.

Proof. Note that the sampling rate of Fleet1 is γ^i for $i \geq 0$. We say that Fleet1 is at level i when the sampling rate is γ^i. For $i \in [0, +\infty)$, define events S_i and B_i as follows. Event B_i: Suppose that we execute algorithm Bern(γ^i), and we have $\left|Z^t_{\gamma^i} - \xi^t\right| \geq \epsilon\xi^t$. S_i is the event that Fleet1 stops at level i.

Define event B: $\left|Y^t_{\text{Fleet1}} - \xi^t\right| \geq \epsilon\xi^t$. We decompose the probability of event B in terms of B_i and S_i.

$$\begin{aligned}\Pr[B] &= \sum_{i=0}^{\infty} \Pr[B_i \wedge S_i] = \sum_{i=0}^{\ell} \Pr[B_i \wedge S_i] + \sum_{i=\ell+1}^{\infty} \Pr[B_i \wedge S_i] \\ &\leq \sum_{i=0}^{\ell} \Pr[B_i] + \sum_{i=\ell+1}^{\infty} \Pr[S_i]\end{aligned}$$

where $\ell = \frac{\log(M/6t)}{\log\gamma}$. The sampling probability at level i is γ^i, by Lemma 3 we have $\sum_{i=0}^{\ell} \Pr[B_i] = \sum_{i=0}^{\ell} 8h^t \cdot exp\left(-\frac{\epsilon^2 \xi^t \gamma^{4i}}{12h^t}\right) = \sum_{i=0}^{\ell} \alpha^{(\gamma^{-4i})}$ where $\alpha = 8h^t \cdot exp\left(-\frac{\epsilon^2 \xi^t \gamma^{4\ell}}{12h^t}\right)$ and $\alpha < 1$.

When $\gamma \leq 0.9$, $\gamma^{-4} > e^{1/e}$, we have $\gamma^{-4i} \geq i$ for any $i \geq 1$. Applying this fact, and using the bound on M we get: $\sum_{i=0}^{\ell} \alpha^{(\gamma^{-4i})} \leq \alpha + \sum_{i=1}^{\ell} \alpha^i \leq \frac{2\alpha}{1-\alpha} \leq \frac{\delta}{2}$

Let X_ℓ denote the number of edges in $\mathcal{R}$ when Fleet1 is at level ℓ. Note that the event Fleet1 stops at level higher than ℓ is equivalent to the event that X_ℓ is greater than the reservoir size M. By $\mathbb{E}[X_\ell] = t \cdot \gamma^\ell$ and Chernoff bound, we have $\sum_{i=\ell+1}^{\infty} \Pr[S_i] = \Pr[X_\ell > M] \leq \Pr[X_\ell > 6 \cdot \mathbb{E}[X_\ell]] \leq \frac{\delta}{2}$. Combining the above bounds, we arrive at $\Pr[B] \leq \delta$. □

4.2 Improved Adaptive: Fleet2 and Fleet3

We present two algorithms Fleet2 and Fleet3 which improve upon Fleet1, providing a better memory-accuracy tradeoff. Fleet2 is similar to Fleet1, but handles sub-sampling differently. Say Fleet1 is at "level i" when its sampling probability is γ^i. In Fleet1, each time the level changes from i to $(i+1)$, edges are discarded according to a random process, and the number of butterflies is recomputed on the new reservoir from scratch (Line 8 of Algorithm 1, shown in pink color). Due to this re-computation, butterflies that were already detected at the higher sampling rate (level i) may no longer have all their edges present at the lower sampling rate (level $(i+1)$). In contrast, Fleet2 does not recompute when the reservoir is sub-sampled. Instead, the current butterfly count at level i is maintained, and as more butterflies are detected at level $i + 1$, they are accumulated into this estimate. It can be expected that Fleet2 obtains a better accuracy than Fleet1, since it "catches" more butterflies than Fleet1. It is easy to see that the estimation of Fleet2 is unbiased. In addition to better accuracy, Fleet2 is also slightly faster than Fleet1, since it avoids recomputation of butterflies at the sub-sampling step.

Fleet3 (described in Algorithm 2) improves accuracy over Fleet1 and Fleet2 by handling new edges differently. This idea is inspired by Algorithm MASCOT of [28], which used the idea in the context of counting triangles from a graph stream (the same idea is also used in [47]). Upon receiving a new edge, the estimate is updated by accounting for butterflies that are created by the new edge (and existing sampled edges), even before deciding whether or not to sample the new edge (see Line 1). In other words, Fleet3 first updates the estimate and then samples. If the current edge sampling probability is p, then the probability of detecting a butterfly involving the new edge increases from p^4 (in Fleet1) to p^3 (in Fleet3). This helps increase the accuracy of butterfly counting while using the same memory. Algorithm 2 has further details. We omit further details and proofs, due to space constraints.

Algorithm 2: FLEET3 ($\mathcal{S}, M$): Adaptive sampling

```
Input: Edge stream $\mathcal{S}$, max. reservoir size $M$, resampling
       parameter $\gamma$ (default value is $\gamma = 0.5$)
Output: Estimate of $\xi^t$, the number of butterflies at $t$
$p \leftarrow 1, \mathcal{R} \leftarrow \emptyset, t \leftarrow 0, \beta \leftarrow 0$
for each edge $e$ in $\mathcal{S}$ do
    $t \leftarrow t + 1$
    $\beta \leftarrow \beta + p^{-3} \times$ BFC-EDGE($e$, $\mathcal{R}$)
    while $|\mathcal{R}| \geq M$ do
        $p \leftarrow \gamma p$
        for each edge $e \in \mathcal{R}$ do
            Keep $e$ in $\mathcal{R}$ with prob. $\gamma$ and discard with prob. $1 - \gamma$
    if coin ($p$) is Head then $\mathcal{R} \leftarrow \mathcal{R} \cup \{e\}$
    $Y^t_{\text{FLEET3}} \leftarrow \beta$
```

Comments: Instead of Bernoulli sampling, we could also use reservoir sampling of edges as the basis. We took the route of Bernoulli sampling to simplify the analysis, since it leads to edges being sampled independent of each other (given a sampling probability), unlike reservoir sampling, where the sampling of edges are not independent events. Our initial implementations of algorithms based on reservoir sampling without replacement, showed that the accuracy-memory tradeoffs were similar to that of our current algorithms.

We can also achieve estimates of per-vertex butterfly counts (the number of butterflies that each vertex is a part of) using a similar sampling approach of estimating per-vertex subgraph counts from the reservoir, and maintaining additional state for each vertex. A detailed study of local butterfly counting is a goal for future work.

5 SLIDING WINDOW STREAMING

In this section, we consider butterfly counting in two types of sliding window models: sequence-based and time-based. We assume the number of edges in a window is (much) greater than the available memory M – otherwise, the entire window can be stored within memory and an exact algorithm can be applied.

5.1 Sequence-based Window (FLEETSSW)

We first present FLEETSSW for sequence-based sliding window (Algorithm 3), which is based on maintaining a sample of edges from within the sliding window. In the initial stages of observation, all edges fit in memory, and as more edges are observed, we recursively decrease the sampling probability as in FLEET1, to ensure that the sample fits in memory. However, once the number of edges in a window reaches W, it will stay at W henceforth. As a result, when the edge sampling probability p reaches M/W, the algorithm does not decrease p any further holds it at M/W[1]. The algorithm stores only active edges in the sample, i.e., any item (e, t') such that t' is not within the current window is discarded.

Let Y^t_{sw} denote an estimate returned by Algorithm 3, of ξ^t_W, the number of butterflies in the window at time t.

Lemma 5. The space of $\mathcal{R}$ in Algorithm 3 is no greater than M in expectation and $\Pr(|\mathcal{R}| \geq 2M) \leq \left(\frac{e}{4}\right)^M$. Y^t_{sw} is an unbiased estimate of ξ^t_W. For parameters $0 < \epsilon < 1$ and $0 < \delta < 1$, when $M \geq 6W \cdot \sqrt[4]{\frac{12h^t}{\epsilon^2 \xi^t_W} \cdot \ln \frac{8h^t(4+\delta)}{\delta}}$, then $\Pr\left[\left|Y^t_{\text{sw}} - \xi^t_W\right| \geq \epsilon \xi^t_W\right] \leq \delta$.

Algorithm 3: FLEETSSW ($\mathcal{S}$,M,W): Seq-based SW

```
Input: Edge stream $\mathcal{S}$, max. reservoir size $M$, window size $W$ ($\gg M$)
Output: Estimate of $\xi^t_W$, the number of butterflies in window at $t$
$p \leftarrow 1, \mathcal{R} \leftarrow \emptyset, t \leftarrow 0, \beta \leftarrow 0$
for each edge $e$ in $\mathcal{S}$ do
    $t \leftarrow t + 1$
    if $p > (M/W)$ then Run FLEET1 ($\mathcal{S}$, $M$) and update $p$, $\beta$, $\mathcal{R}$
    else
        $p \leftarrow (M/W)$
        if coin ($p$) is Head then
            $\mathcal{R} \leftarrow \mathcal{R} \cup \{e, t\}$, and $\beta \leftarrow \beta + p^{-4} \times$ BFC-EDGE($e$, $\mathcal{R}$)
    /* Delete expired edges and update estimate */
    if $(e', t') \in \mathcal{R}$ s.t. $t' \leq (t - W)$ then
        $\beta \leftarrow \beta - p^{-4} \times$ BFC-EDGE($e'$, $\mathcal{R}$), and $\mathcal{R} \leftarrow \mathcal{R} \setminus (e', t')$
    $Y^t_{\text{sw}} \leftarrow \beta$
```

Proof. We sketch the proof ideas and omit details, due to space constraints. For the space complexity, note that when $p > M/W$, the algorithm runs FLEET1 and its space is strictly bounded by M. Otherwise, $p = M/W$ and the number of edges in the sample is a binomial random variable with parameters W and M/W, and the space bounds follow using Chernoff bounds.

At any given time, each edge currently in the window is sampled into $\mathcal{R}$ with probability p, and $\mathbb{E}\left[Y^t_{\text{sw}}\right] = \xi^t_W \cdot p^4$.When $p > (M/W)$, we apply results from FLEET1 for expectation (Lemma 2) and concentration (Lemma 4) to show the corresponding properties of Y^t_{sw}.

When $p = (M/W)$, we rely on an analysis similar to Algorithm BERN in Lemma 3 and derive the concentration result that $\Pr\left[\left|Y^t_{\text{sw}} - \xi^t_W\right| \geq \epsilon \xi^t_W\right] \leq 8h^t \cdot exp\left(-\frac{\epsilon^2 \xi^t_W}{12h^t} \cdot \left(\frac{M}{W}\right)^4\right) \leq \delta$. □

5.2 Time-based Sliding Window (FLEETTSW)

We next consider the case of a time-based sliding window, where each element has an associated timestamp, and the window at time t consists of all elements with timestamps greater than $(t - W)$, where W is a specified window size. Handling a time-based sliding window is more challenging than a sequence-based window since the number of elements within a time-based window can grow and shrink with time. The sequence-based window can be seen as a special case of time-based window such that at each time, exactly one edge arrives in the stream.

FLEETTSW (Algorithm 4), our algorithm for time-based sliding window, can estimate the number of butterflies when the window size W is provided at query time. We assume an upper bound n_{max} number of edges within a window. Let $T = \lceil 1 + \log_\gamma \frac{M}{n_{max}} \rceil$. FLEETTSW is based on maintaining not only a single sample, as in FLEETSSW or FLEET1, but T reservoirs $\mathcal{R}_i$, $i = 0, 1, 2, \ldots$, at different sampling rates. Every edge is sampled into $\mathcal{R}_0$. For $i > 0$, each edge that was sampled into $\mathcal{R}_{i-1}$ is sampled into $\mathcal{R}_i$ with probability γ. Each reservoir has a capacity of $M' = M/T$ edges, and contains the most recent edges sampled into the reservoir. Each $\mathcal{R}_i$ is stored as a first-in-first-out queue, so that if a new edge enters when the queue is full, the edge with the earliest timestamp is deleted.

Lemma 6. Y^t_{tw} is an unbiased estimate of ξ^t_W. If $M \in \Omega\left(\log_\gamma \frac{M}{n_{\max}} \sqrt[4]{\frac{n^4_{\max} h^t}{\epsilon^2 \xi^t_W} \ln \frac{h^t}{\delta}}\right)$, so $\Pr\left[|Y^t_{\text{tw}} - \xi^t_W| \geq \epsilon \xi^t_W\right] \leq \delta$.

[1] For simplicity of exposition, we assume M/W is a power of γ.

Algorithm 4: FLEETTSW ($\mathcal{S}$, M, n_{max}): Time-based SW

```
Input: Edge stream $\mathcal{S}$, reservoir size $M$, $n_{max}(\gg M)$ (the
       maximum number of elements within a window)
Output: Estimate of $\xi(G_W^t)$
$T \leftarrow \lceil 1 + \log_\gamma \frac{M}{n_{max}} \rceil$
// $d_\ell$ is the time of most recent discarded edge in $\mathcal{R}_\ell$
$\forall i \le T,\ \mathcal{R}_i \leftarrow \emptyset,\ d_i \leftarrow 0$
for each edge $e$ in $\mathcal{S}$ at time $t$ do
    $\ell \leftarrow 0$
    do
        $R_\ell \leftarrow R_\ell \cup \{(e, t)\}$
        if $|R_\ell| > \frac{M}{T}$ then
            $R_\ell \leftarrow R_\ell \setminus \{(e^*, t^*)\}$ s.t. $t^* = \min\{t' \mid (e', t') \in R_\ell\}$
            $d_\ell \leftarrow t^*$
        $\ell \leftarrow \ell + 1$
    while coin $(\gamma)$ is Head
/* Upon a query window size $W$ at time $c$ */
$\ell_* \leftarrow argmin_{\ell \in [T]}\{d_\ell \mid d_\ell \le (c - W)\}$
$\mathcal{A} \leftarrow \{(e, t') \in \mathcal{R}_{\ell_*}\}$ s.t. $t' > (c - W)$ // sample of window
$Y_{\text{tw}}^t \leftarrow \gamma^{-4\ell_*} \times \xi(\mathcal{A})$
```

Proof. At query time t, when presented with a window size W, let G_W^t denote the graph consisting of all edges that have timestamps in $[t - W + 1, t]$. For level $\ell \in \{0, 1, 2, \ldots\}$, let $G_W^t(\ell)$ be defined inductively as follows. $G_W^t(0) = G_W^t$. For $i \ge 0$, $G_W^t(i+1)$ is derived from $G_W^t(i)$ by choosing each edge in $G_W^t(i)$ with probability γ. We note that in Algorithm 4, $\mathcal{R}_i$ contains the M/T most recent elements from $G_W^t(i)$. Further, when a query arrives at time t, the algorithm uses $\mathcal{R}_{\ell_*}$ such that ℓ_* is the smallest value, where $G_W^t(\ell_*)$ is completely contained in $\mathcal{R}_{\ell_*}$.

Let $Z_W^t(i)$ denote the estimate of $\xi(G_W^t)$ derived from $\mathcal{R}_i$. Similar to the proof of Lemma 4, we define: event S_i is the event that the algorithm chooses $\mathcal{R}_i$ to answer the sliding window query, and B_i is the event that the estimate $Z_W^t(i)$ has a relative error that is greater than ϵ. The probability that the algorithm fails to return an estimate that has a relative error within ϵ is given by $\Pr[B] = \sum_{i=0}^{\infty} \Pr[S_i \wedge B_i]$. Using an argument similar to the proof of Lemma 4, we arrive at the result. □

6 EXPERIMENTAL EVALUATION

We experimentally evaluate the infinite window and sliding window algorithms on real-world temporal bipartite networks with hundreds of millions of edges from a variety of domains, such as social, web, and rating networks.

Networks and experimental setup: We used five real-world temporal bipartite networks from the publicly available KONECT repository [26], summarized in Table 1.[2] `Movie-lens` is the ratings by users for movies. `Edit-frwiki` is a bipartite network of editors and pages of the French Wikipedia where each edge represents an edit. `Edit-enwiki` is the English version of `Edit-frwiki`. `Yahoo-song` is a ratings by users for songs. `Bag-pubmed` is a word-document bipartite network. Note that `Bag-pubmed` is not a temporal network, and we generated a stream by randomly permuting the edge set of `Bag-pubmed`. `Edit-frwiki`, `Edit-enwiki`, and `Bag-pubmed` had multiple edges between the same node pairs, and we only considered the first interaction. Edges are read from the stream in the order of timestamps. Figure 3 shows the number of butterflies as a function of stream size.

Graphs	|E|	|V| (Left)	|V| (Right)	ξ	Butterfly density
`Movie-lens`	10 000 054	69 878	10 677	1.1T	1.1×10^{-16}
`Edit-frwiki`	22 090 703	288 275	3 992 426	601.2B	2.5×10^{-18}
`Yahoo-song`	256 804 235	1 000 990	624 961	101.4T	2.3×10^{-20}
`Edit-enwiki`	122 075 170	3 819 691	21 416 395	2T	9.1×10^{-21}
`Bag-pubmed`	483 450 157	8 200 000	141 043	40.8T	7.4×10^{-22}

Table 1: Properties of the bipartite graphs. $|E|$ is the number of edges, ξ the total number of butterflies, and the butterfly density is the ratio $\xi/|E|^4$.

All streaming algorithms were implemented in `C++` and compiled with `g++` compiler using `-O3` as the optimization level. The source code is publicly available at [1]. We run the experiments on a machine equipped with a 2.0 GHz 16-Core Intel E5 2650 processor and 128GB of memory.

6.1 Accuracy

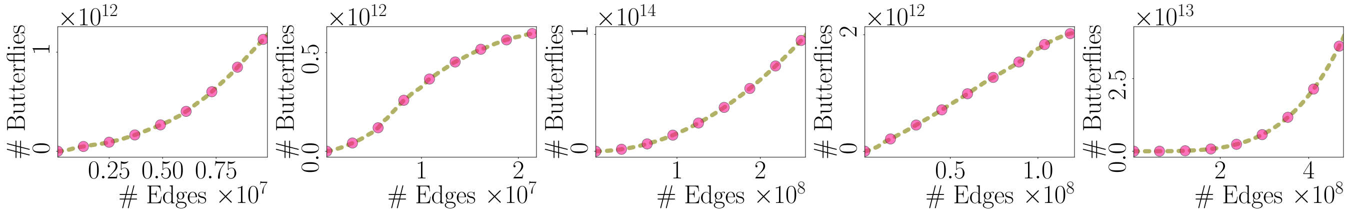
If the true value of the butterfly count is $x > 0$, then the relative error of an estimate $\hat{x}$ is defined as $|x - \hat{x}|/x$ and is usually shown as percent error. We also used MAPE (Mean Average Percentage Error) to measure the accuracy over the entire stream, defined as the average of the relative error, taken over the entire stream.

Figure 4 shows the accuracy on the entire stream vs the reservoir size. Larger reservoirs yield better accuracies, as expected. FLEET3 can keep the estimation error around 1% for all networks by storing only 600K edges in the reservoir. This corresponds to 6%, 2.7%, 0.49%, 0.23%, and 0.12% of the total stream sizes for `Movie-lens`, `Edit-frwiki`, `Edit-enwiki`, `Yahoo-song`, and `Bag-pubmed`, respectively. When the reservoir size is 300K, FLEET3 yields 3% error for `Edit-enwiki` and `Bag-pubmed` and less than 1% for other networks. As expected FLEET2 has better accuracy than FLEET1, and FLEET3 has the best accuracy.

Figure 5 shows the relative error at different points in the stream, for a fixed reservoir size. As the stream size increases, the error of FLEET1 and FLEET2 increase slightly. This can be attributed to the fact that the edge sampling probability p is proportional to $1/t$, where t is the number of edges, and from Lemma 3, the probability of a given relative error decreases with $p^4\xi^t$. Unless ξ^t increases as the fourth power of t, the probability of a given relative error increases with the stream size.

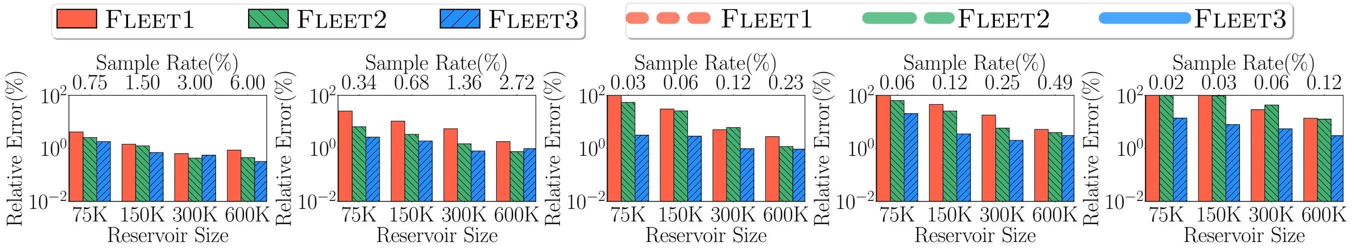
Butterfly Density: We note the errors of FLEET1 and FLEET2 for a given reservoir size are roughly correlated with the butterfly density (ξ^t/t^4 where t is the number of edges). One reason is as follows. Following Lemma 3, the probability of a high relative error decreases with $p^4\xi^t$. Setting $p \approx M/t$ where M is the reservoir size, this is M^4 times the butterfly density $\frac{\xi^t}{t^4}$, showing that the error probability decreases quickly as the butterfly density increases. When the networks are ordered according to increasing butterfly density, we get the order `Bag-pubmed`, `Edit-enwiki`, `Yahoo-song`, `Edit-frwiki`, and `Movie-lens`. We note that for the same reservoir size, this is exactly the increasing order of accuracy (decreasing order or error) for algorithms FLEET1 and FLEET2 (Figure 5). The trend is not so clear for algorithm FLEET3, since its accuracy depends heavily on the temporal order of the edges within a butterfly.

[2] http://konect.uni-koblenz.de/



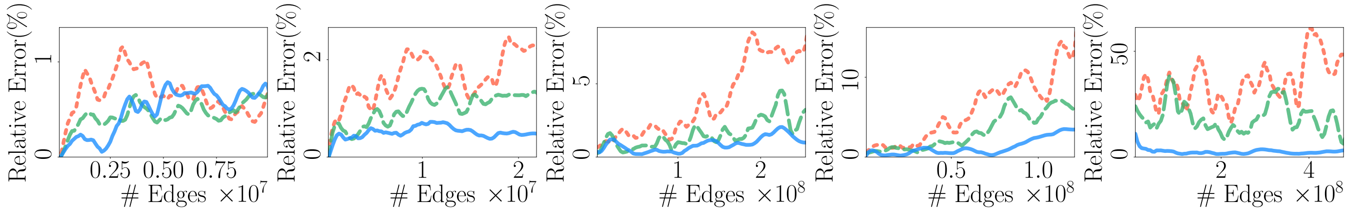
(a) Movie-lens (b) Edit-frwiki (c) Yahoo-song (d) Edit-enwiki (e) Bag-pubmed

Figure 3: Number of butterflies as a function of stream size.



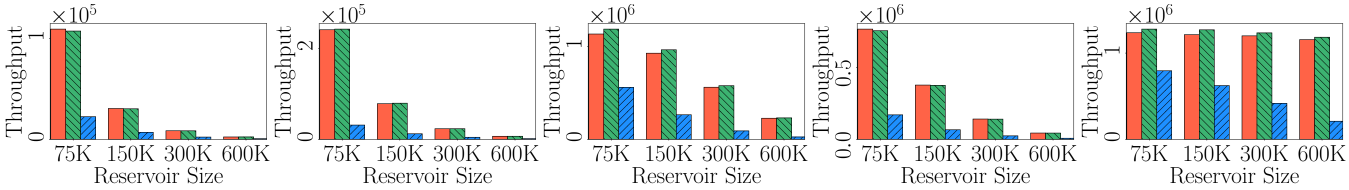
(a) Movie-lens (b) Edit-frwiki (c) Yahoo-song (d) Edit-enwiki (e) Bag-pubmed

Figure 4: Accuracy of Fleet1, Fleet2, and Fleet3 for $\gamma = 0.5$ versus reservoir size. Bottom x-axis shows the reservoir size and top x-axis shows the sample rate, defined as the ratio of the reservoir size to the stream size.



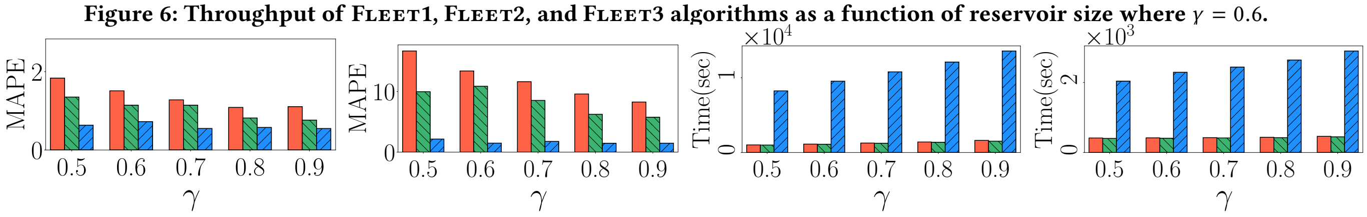
(a) Movie-lens (b) Edit-frwiki (c) Yahoo-song (d) Edit-enwiki (e) Bag-pubmed

Figure 5: Accuracy of Fleet1, Fleet2, and Fleet3 at different points in the stream, reservoir size is $300K$ and $\gamma = 0.9$.



(a) Movie-lens (b) Edit-frwiki (c) Yahoo-song (d) Edit-enwiki (e) Bag-pubmed

Figure 6: Throughput of Fleet1, Fleet2, and Fleet3 algorithms as a function of reservoir size where $\gamma = 0.6$.



(a) Yahoo-song (error vs. γ) (b) Bag-pubmed (error vs. γ) (c) Yahoo-song (runtime vs. γ) (d) Bag-pubmed (runtime vs. γ)

Figure 7: Accuracy and runtime of Fleet1, Fleet2, and Fleet3 as a function of γ where $M = 600K$.

6.2 Runtime and Throughput

The better accuracy of Fleet3 comes at the cost of increased runtime. From Figure 6, we see Fleet3 has the lowest throughput (number of edges processed per second), while Fleet1 and Fleet2 have similar throughputs. The reason is there is one per-edge butterfly computation for each arriving edge in Fleet3, where as there is one such computation only for each sampled edge in Fleet1 and Fleet2. The throughput decreases as the reservoir size increases, due to the increased cost of per-edge butterfly counting on the reservoir. Fleet3 is able to achieve quite a high throughput, e.g. 6.2×10^5 edges per second on graph Bag-pubmed with reservoir size 150K, making it suitable for practical scenarios. Fleet2 always

Graphs	Fleet1	Fleet2	Fleet3	GPS [3]	BC [9]
Movie-lens	1.03	0.72	**0.69**	89.32	103.23
Edit-frwiki	4.37	1.7	**1.68**	63.92	102.36
Yahoo-song	13.97	5.62	**0.78**	43.1	104.48
Edit-enwiki	19.43	9.94	**2.95**	46.65	85.38
Bag-pubmed	103.74	91.59	**5.65**	14.22	113.77

Table 2: MAPE (Mean Absolute Percent Error) of different algorithms for $\gamma = 0.8$ and $M = 150$K

has a slightly higher throughput than Fleet1. Overall, Fleet3 has the best accuracy with a good throughput, while Fleet2 trades a lower accuracy for a higher throughput.

6.3 Impact of γ on runtime and accuracy

Figures 7a and 7b show the accuracy as a function of γ. As γ increases, the average size of the reservoir increases, while the frequency of sub-sampling also increases. The accuracies of Fleet1 and Fleet2 improve slightly as γ increases from 0.5 to 0.9 e.g. for graph Yahoo-song. In contrast, from Figures 7c and 7d, the runtime increases for all estimators as γ increases. A value of γ of about 0.7 seems to be a good "middle ground" since it achieves nearly the best throughput as well as accuracy.

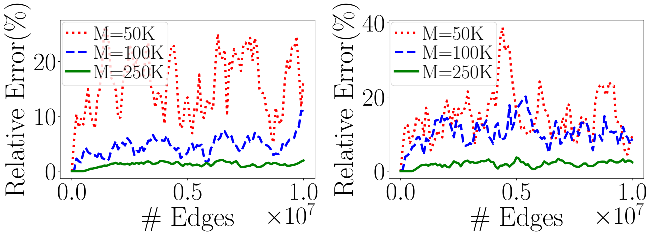
6.4 Comparison with prior work

In this section, we present a comparison between our methods and prior works, including [3, 9, 24, 31].

Table 2 presents a comparison with methods: Graph Priority Sampling (GPS)[3] and the work of Bera and Chakrabarti (BC) (Algorithm 1 from Section 3.1 of [9]) for a reservoir size of $150K$ (we found similar results for other reservoir sizes, ranging from 75K to $600K$). GPS is a subgraph counting algorithm based on a weighted sample of edges, which we specialized for the case of butterfly estimation. We observe that Fleet3 significantly outperforms GPS on all networks, and Fleet1 and Fleet2 outperform GPS on all networks except Bag-pubmed. Since GPS stores additional information for each edge (*weight* and *rank* – see [3] for details), we stored $75K$ edges in the reservoir for GPS to keep its memory equal to our algorithms. The results were quite similar even if we gave twice the memory to GPS. If we used a sample of $150K$ edges in GPS, its error ranged from 7.78% (Bag-pubmed) to 90% (Movie-lens) still much worse than our algorithms, especially Fleet3.

We compared with BC while holding memory equal, even though BC is a two-pass streaming algorithm, which cannot be modified to work in a single pass, and works under a more powerful computational model than our algorithms. With a reservoir of size $150K$, we could run $75K$ basic estimators of BC, each of which maintained a sample of two edges. On all streams, all of our algorithms outperformed BC by significant margins.

We implemented the algorithm of Manjunath et al. [31], which estimates the number of cycles in a stream using sketches based on complex random variables. To the best of our knowledge, we are the first to implement this algorithm, and even the authors of [31] have not provided an implementation. The accuracy of [31] is very poor – the error of the estimator is more than 100% for both graph streams Yahoo-song and Bag-pubmed, even with a memory of 600K estimators. In addition, we found their algorithm slow and impractical. The reason is that for each arriving edge in a stream, the algorithm needs to update the values of many complex valued sketches, where the number of sketches is as large as the reservoir size. The throughput of [31] on both graphs is only 5.9 edges per second, $\approx 21K$ edges in one hour, meaning that it is about 10^5 times slower than algorithm Fleet3 (see throughput of Fleet3 in Figures 6c and 6e). The algorithms of [24] are not practical for handling large graph streams, since they follows a similar structure (this has not been implemented either, to the best of our knowledge).



(a) Movie-lens

(b) Yahoo-song

Figure 8: Relative error vs. number of edges received for FleetSSW. Window size = 5×10^6 edges, $\gamma = 0.9$.

6.5 Sliding Window

Figure 8 shows the relative error of FleetSSW for window size $W = 5$M edges, when the reservoir size is varied from 1% to 5% of W. The accuracy improves as the reservoir size increases; when M is 5% of the window size, the relative error is always under 5%. The number of butterflies within a window ranges from 5×10^{10} to 10^{11} for Movie-lens and 1×10^{10} to 6×10^{10} for Yahoo-song. We also experimented with FleetTSW for time-based windows. We used 30 queries, and a window size is randomly generated at query time. When the reservoir size M is 10% of the stream size, the average relative error over the queries is 2.55% for movie, 5.52% for Edit-frwiki and 6.32% for Edit-enwiki. This result shows FleetTSW can achieve good accuracy using memory much smaller than the whole stream.

7 CONCLUSION

We presented a lower bound as well as one-pass streaming algorithms for estimating the number of butterflies from a bipartite graph stream. While our lower bound rules out space-efficient algorithms that are accurate on all graph streams, it leaves open the possibility of space-efficient algorithms for graph streams where the number of butterflies is large, such as in every real-world graph stream that we tried. Our algorithms Fleet1, Fleet2, and Fleet3 are based on adaptive random sampling from the graph stream, achieve high accuracy on real-world streams, and are backed by rigorous theoretical guarantees. We also presented algorithms FleetSSW and FleetTSW for sequence-based and time-based sliding windows respectively. This work is one of the first to explore streaming motif counting on bipartite graphs, and leads to many follow-up questions. (1) Extensions to general motif counting on bipartite graph streams (2) Can we combine the benefits of improved accuracy as in Fleet3 with the faster runtime of Fleet2? (3) Algorithms for multi-pass and external memory models.

8 ACKNOWLEDGMENT

The work of SS, YZ, and ST is supported in part by the National Science Foundation through grants 1527541 and 1725702.

REFERENCES

[1] 2019. Source Code. https://github.com/beginner1010/fleet.

[2] Gagan Aggarwal, Yang Cai, Aranyak Mehta, and George Pierrakos. 2014. Biobjective online bipartite matching. In *International Conference on Web and Internet Economics*. Springer, 218–231.

[3] Nesreen K Ahmed, Nick Duffield, Theodore L Willke, and Ryan A Rossi. 2017. On sampling from massive graph streams. *Proceedings of the VLDB Endowment* 10, 11 (2017), 1430–1441.

[4] Nesreen K Ahmed, Jennifer Neville, Ryan A Rossi, and Nick Duffield. 2015. Efficient graphlet counting for large networks. In *2015 IEEE International Conference on Data Mining*. IEEE, 1–10.

[5] Sinan G Aksoy, Tamara G Kolda, and Ali Pinar. 2017. Measuring and modeling bipartite graphs with community structure. *Journal of Complex Networks* 5, 4 (2017), 581–603.

[6] Noga Alon, Raphael Yuster, and Uri Zwick. 1997. Finding and counting given length cycles. *Algorithmica* 17, 3 (1997), 209–223.

[7] B. Babcock, M. Datar, and R. Motwani. 2002. Sampling from a Moving Window over Streaming Data. In *SODA*.

[8] Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides Gionis. 2010. Efficient algorithms for large-scale local triangle counting. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 4, 3 (2010), 13.

[9] Suman K Bera and Amit Chakrabarti. 2017. Towards tighter space bounds for counting triangles and other substructures in graph streams. In *34th Symposium on Theoretical Aspects of Computer Science (STACS 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[10] Ilaria Bordino, Debora Donato, Aristides Gionis, and Stefano Leonardi. 2008. Mining large networks with subgraph counting. In *2008 Eighth IEEE International Conference on Data Mining*. IEEE, 737–742.

[11] Vladimir Braverman, Rafail Ostrovsky, and Dan Vilenchik. 2013. How hard is counting triangles in the streaming model?. In *International Colloquium on Automata, Languages, and Programming*. Springer, 244–254.

[12] Vladimir Braverman, Rafail Ostrovsky, and Carlo Zaniolo. 2009. Optimal sampling from sliding windows. In *Proceedings of the twenty-eighth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, 147–156.

[13] Laurent Bulteau, Vincent Froese, Konstantin Kutzkov, and Rasmus Pagh. 2016. Triangle counting in dynamic graph streams. *Algorithmica* 76, 1 (2016), 259–278.

[14] Luciana S Buriol, Gereon Frahling, Stefano Leonardi, and Christian Sohler. 2007. Estimating clustering indexes in data streams. In *European Symposium on Algorithms*. Springer, 618–632.

[15] Xiaowei Chen and John Lui. 2017. A unified framework to estimate global and local graphlet counts for streaming graphs. In *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017*. ACM, 131–138.

[16] Graham Cormode and Hossein Jowhari. 2017. A second look at counting triangles in graph streams (corrected). *Theoretical Computer Science* 683 (2017), 22–30.

[17] Hongbo Deng, Michael R Lyu, and Irwin King. 2009. A generalized co-hits algorithm and its application to bipartite graphs. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 239–248.

[18] Fazle E Faisal and Tijana Milenković. 2014. Dynamic networks reveal key players in aging. *Bioinformatics* 30, 12 (2014), 1721–1729.

[19] Phillip B Gibbons and Srikanta Tirthapura. 2002. Distributed streams algorithms for sliding windows. In *Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures*. ACM, 63–72.

[20] András Hajnal and Endre Szemerédi. 1970. Proof of a conjecture of P. Erdos. *Combinatorial theory and its applications* 2 (1970), 601–623.

[21] Guyue Han and Harish Sethu. 2017. Edge sample and discard: A new algorithm for counting triangles in large dynamic graphs. In *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017*. ACM, 44–49.

[22] Madhav Jha, C Seshadhri, and Ali Pinar. 2015. Path sampling: A fast and provable method for estimating 4-vertex subgraph counts. In *Proceedings of the 24th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 495–505.

[23] Hossein Jowhari and Mohammad Ghodsi. 2005. New streaming algorithms for counting triangles in graphs. In *International Computing and Combinatorics Conference*. Springer, 710–716.

[24] Daniel M Kane, Kurt Mehlhorn, Thomas Sauerwald, and He Sun. 2012. Counting arbitrary subgraphs in data streams. In *International Colloquium on Automata, Languages, and Programming*. Springer, 598–609.

[25] Mihail N Kolountzakis, Gary L Miller, Richard Peng, and Charalampos E Tsourakakis. 2012. Efficient triangle counting in large graphs via degree-based vertex partitioning. *Internet Mathematics* 8, 1-2 (2012), 161–185.

[26] Jérôme Kunegis. 2013. Konect: the koblenz network collection. In *Proceedings of the 22nd International Conference on World Wide Web*. ACM, 1343–1350.

[27] Lin Li, Zhenglu Yang, Ling Liu, and Masaru Kitsuregawa. 2008. Query-URL Bipartite Based Approach to Personalized Query Recommendation.. In *AAAI*, Vol. 8. 1189–1194.

[28] Yongsub Lim and U Kang. 2015. Mascot: Memory-efficient and accurate sampling for counting local triangles in graph streams. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 685–694.

[29] Pedro G Lind, Marta C Gonzalez, and Hans J Herrmann. 2005. Cycles and clustering in bipartite networks. *Physical review E* 72, 5 (2005), 056127.

[30] Boge Liu, Long Yuan, Xuemin Lin, Lu Qin, Wenjie Zhang, and Jingren Zhou. 2019. Efficient (a, β)-core Computation: an Index-based Approach. In *The World Wide Web Conference*. ACM, 1130–1141.

[31] Madhusudan Manjunath, Kurt Mehlhorn, Konstantinos Panagiotou, and He Sun. 2011. Approximate counting of cycles in streams. In *European Symposium on Algorithms*. Springer, 677–688.

[32] Aranyak Mehta. 2013. Online matching and ad allocation. *Foundations and Trends® in Theoretical Computer Science* 8, 4 (2013), 265–368.

[33] Tijana Milenković and Nataša Pržulj. 2008. Uncovering biological network function via graphlet degree signatures. *Cancer informatics* 6 (2008), CIN–S680.

[34] Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. 2002. Network motifs: simple building blocks of complex networks. *Science* 298, 5594 (2002), 824–827.

[35] Rasmus Pagh and Charalampos E Tsourakakis. 2012. Colorful triangle counting and a mapreduce implementation. *Inform. Process. Lett.* 112, 7 (2012), 277–281.

[36] A Pavan, Kanat Tangwongsan, Srikanta Tirthapura, and Kun-Lung Wu. 2013. Counting and sampling triangles from a graph stream. *Proceedings of the VLDB Endowment* 6, 14 (2013), 1870–1881.

[37] Georgios A Pavlopoulos, Panagiota I Kontou, Athanasia Pavlopoulou, Costas Bouyioukos, Evripides Markou, and Pantelis G Bagos. 2018. Bipartite graphs in systems biology and medicine: a survey of methods and applications. *GigaScience* 7, 4 (2018), giy014.

[38] Ali Pinar, C Seshadhri, and Vaidyanathan Vishal. 2017. Escape: Efficiently counting all 5-vertex subgraphs. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1431–1440.

[39] Nataša Pržulj. 2007. Biological network comparison using graphlet degree distribution. *Bioinformatics* 23, 2 (2007), e177–e183.

[40] Garry Robins and Malcolm Alexander. 2004. Small worlds among interlocking directors: Network structure and distance in bipartite graphs. *Computational & Mathematical Organization Theory* 10, 1 (2004), 69–94.

[41] Seyed-Vahid Sanei-Mehri, Ahmet Erdem Sariyuce, and Srikanta Tirthapura. 2018. Butterfly Counting in Bipartite Networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2150–2159.

[42] Ahmet Erdem Sarıyüce and Ali Pinar. 2018. Peeling bipartite networks for dense subgraph discovery. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*. ACM, 504–512.

[43] Doris Schiöberg, Fabian Schneider, Stefan Schmid, Steve Uhlig, and Anja Feldmann. 2015. Evolution of directed triangle motifs in the google+ osn. *arXiv preprint arXiv:1502.04321* (2015).

[44] Jessica Shi and Julian Shun. 2019. Parallel Algorithms for Butterfly Computations. *arXiv preprint arXiv:1907.08607* (2019).

[45] Kijung Shin, Jisu Kim, Bryan Hooi, and Christos Faloutsos. 2018. Think before you discard: Accurate triangle counting in graph streams with deletions. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 141–157.

[46] Omkar Singh, Kunal Sawariya, and Polamarasetty Aparoy. 2014. Graphlet signature-based scoring method to estimate protein–ligand binding affinity. *Royal Society open science* 1, 4 (2014), 140306.

[47] Lorenzo De Stefani, Alessandro Epasto, Matteo Riondato, and Eli Upfal. 2017. Triest: Counting local and global triangles in fully dynamic streams with fixed memory size. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 11, 4 (2017), 43.

[48] Kanat Tangwongsan, Aduri Pavan, and Srikanta Tirthapura. 2013. Parallel triangle counting in massive streaming graphs. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*. ACM, 781–786.

[49] Ata Turk and Duru Turkoglu. 2019. Revisiting Wedge Sampling for Triangle Counting. In *The World Wide Web Conference*. ACM, 1875–1885.

[50] Jia Wang, Ada Wai-Chee Fu, and James Cheng. 2014. Rectangle counting in large bipartite graphs. In *2014 IEEE International Congress on Big Data*. IEEE, 17–24.

[51] Kai Wang, Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang. 2019. Vertex priority based butterfly counting for large-scale bipartite networks. *Proceedings of the VLDB Endowment* 12, 10 (2019), 1139–1152.

[52] Pinghui Wang, Yiyan Qi, Yu Sun, Xiangliang Zhang, Jing Tao, and Xiaohong Guan. 2017. Approximately counting triangles in large graph streams including edge duplicates with a fixed memory usage. *Proceedings of the VLDB Endowment* 11, 2 (2017), 162–175.

[53] R. Kumar Z. Bar-Yossef and D. Sivakumar. 2002. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proc. SODA*.

[54] Rong Zhu, Zhaonian Zou, and Jianzhong Li. 2018. Fast Rectangle Counting on Massive Networks. In *2018 IEEE International Conference on Data Mining (ICDM)*. IEEE, 847–856.