Butterfly Counting in Bipartite Networks

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

Ahmet Erdem Sarıyüce
University at Buffalo
Buffalo, New York
erdem@buffalo.edu

Srikanta Tirthapura
Iowa State University
Ames, Iowa
snt@iastate.edu

ABSTRACT
We consider the problem of counting motifs in bipartite affiliation networks, such as author-paper, user-product, and actor-movie relations. We focus on counting the number of occurrences of a "butterfly", a complete 2 × 2 biclique, the simplest cohesive higher-order structure in a bipartite graph. Our main contribution is a suite of randomized algorithms that can quickly approximate the number of butterflies in a graph with a provable guarantee on accuracy. An experimental evaluation on large real-world networks shows that our algorithms return accurate estimates within a few seconds, even for networks with trillions of butterflies and hundreds of millions of edges.

ACM Reference Format:

1 INTRODUCTION
Graph motifs are used to model and examine interactions among small sets of vertices in networks. Finding frequent patterns of interactions can reveal functions of participating entities [3, 7, 11, 12, 22, 27] and help characterize the network. Also known as graphlets or higher-order structures, motifs are regarded as basic building blocks of complex networks in domains such as social networks, food webs, and neural networks [16]. For this reason, finding and counting motifs are among the most important and widely used network analysis procedures. The triangle is the most basic motif in a unipartite network, and graph mining literature is abundant with triangle counting algorithms for stationary networks [27, 29] as well as dynamic networks [8, 14, 21]. There are also studies that consider structures with more than 3 vertices [11, 12, 22], but these primarily focus on cliques, such as 4-cliques and 5-cliques.

In this work, we focus on bipartite (affiliation) networks, an important type of network for many applications [6, 13]. For example, relationships between authors and papers can be modeled as a bipartite graph, where authors form one vertex partition, papers form the other vertex partition, and an author has an edge to each paper that she published. Other examples include user-product relations, word-document affiliations, and actor-movie networks. Bipartite graphs can represent hypergraphs that capture many-to-many relations among entities. A hypergraph \( H = (V_H, E_H) \) with vertex set \( V_H \) and edge set \( E_H \), where each hyper-edge \( h \in E_H \) is a set of vertices, can be represented as a bipartite graph with vertex set \( V_H \cup E_H \) with one partition for \( V_H \) and another for \( E_H \), and an edge from a vertex \( v \in V_H \) to an edge \( h \in E_H \) if \( v \) is a part of \( h \) in \( H \). For example, a hypergraph corresponding to an author-paper relation where each paper is associated with a set of authors can be represented using a bipartite graph with one partition for authors and one for papers.

A common approach to handle a bipartite network is to reduce it to a unipartite co-occurrence network by a projection [17, 18]. A projection selects a vertex partition as the set of entities, and creates a unipartite network whose vertex set is the set of all entities and where two entities are connected if they share an affiliation in the bipartite network. For the author-paper network, a projection on the authors creates a unipartite co-authorship network. However, such a projection causes the number of edges in the graph to explode, artifically boosts the number of triangles and clustering coefficients, and results in information loss. For instance, we observed up to 4 orders of magnitude increase in size when the bipartite network between wikipedia articles and their editors in French is projected onto a unipartite network of articles - number of edges goes from 22M to more than 200B. As a result, it is preferable to analyze bipartite networks directly.

While there is extensive work on motif counting in unipartite networks, these do not apply to bipartite networks. Motifs in bipartite networks are very different from motifs in a unipartite network. The most commonly studied motifs in a unipartite network are cliques of small sizes, but a bipartite graph does not have any cliques with more than two vertices, not even a triangle! Instead, natural motifs in a bipartite network are bicliques of small size.

The most basic motif that models cohesion in a bipartite network is the complete 2 × 2 biclique, also known as a butterfly [4, 26] or a rectangle [31]. Although there have been attempts at defining other cohesive motifs in bipartite networks, such as the complete 3 × 3 biclique [6] and 4-path [19], the butterfly remains the smallest unit of cohesion, and has been used in defining basic metrics such as the clustering coefficient in a bipartite graph [15, 23]. In particular, it is the smallest subgraph that has multiple vertices on each side with multiple common neighbors. It can be considered as playing the same role in bipartite networks as the triangle did in unipartite networks - a building block for community structure. We aim to derive methods which can accurately estimate the number of butterflies in a given large bipartite graph. We view butterfly counting as a first, but important step towards general methods for motif counting and analysis of bipartite affiliation networks.
Contributions: We present fast algorithms to accurately estimate the number of butterflies in a bipartite network. Our algorithms are simple to implement, backed up by theoretical guarantees, and have good practical performance.

- **Exact Butterfly Counting**: We first present an efficient exact algorithm, **ExactBFC**, for counting the number of butterflies in a network. We use a simple measure, the sum of the squares of vertex degrees, to choose which vertex partition of the bipartite graph to start the algorithm from. Leveraging the imbalance between vertex partitions yields significant speedups over the state-of-the-art [31].

- **Randomized Approximate Butterfly Counting**: We introduce efficient randomized algorithms to find the approximate number of butterflies in a network by sampling. Our algorithms are able to derive accurate estimates with error as low as 1% within a few seconds, are much faster than the exact algorithms, and have an insignificant memory print. We present two types of randomized algorithms:

  - **One-shot Sparsification** techniques assume that the entire graph is available for processing. They thin-down the graph to a much smaller sparsified graph through choosing each edge of the original graph with a certain randomized procedure. Exact butterfly counting is then applied on the sparsified graph to estimate the number of butterflies in the original graph. We present two such algorithms — **ESpar** and **CLRSPAR**.

  - **Local Sampling** algorithms, on the other hand, can work under limited access to the input graph. They randomly sample small subgraphs local to an element of the graph, and use them to compute an estimate. This is in contrast to sparsification, which needs a global view of the graph. We investigate sampling of subgraphs localized around a vertex (**VSAMP**), edge (**ESAMP**), and a wedge (**WSAMP**). Sampling algorithms are especially useful when there is a rate-limited API that provides random samples, such as the GNIP framework for Twitter [1] and the Graph API of Facebook [2]. In the rest of this paper, when we say "sampling algorithms", we mean local sampling algorithms.

- **Provable Guarantees**: We prove that the randomized algorithms yield estimates that are equal to the actual number of butterflies in the graph, in expectation. Through a careful analysis of their variance, we show that it is possible to reduce the estimation error to any desired level, through independent repetitions (sampling) or through an appropriate choice of parameters (sparsification).

- **Experiments on real-world networks**: We present results of an evaluation of our algorithms on large real-world networks. These results show that the algorithms can handle massive graphs with hundreds of millions of edges and trillions of butterflies. Our most efficient sampling algorithm, which we call **ESAMP + FAST-BFC**, gives estimates with a relative error less than 1% within 5 seconds, even for large graphs with trillions of butterflies. On such large graphs, (exact) algorithms from prior work took tens of thousands of seconds. We observed a similar behavior with our best one-shot sparsification algorithm, **ESpar**, which typically yields estimates with error less than 1 percent, within 4 secs.

2 PRELIMINARIES

We consider simple, unweighted, bipartite graphs, where there are no self-loops or multiple edges between vertices. Let $G = (V, E)$ be a simple bipartite graph with $n = |V|$ vertices and $m = |E|$ edges. Vertex set $V$ is partitioned into two sets $L$ and $R$ such that $V = L \cup R$ and $L \cap R = \emptyset$. The edge set $E \subseteq L \times R$. For $v \in V$, $\Gamma_v = \{u | (v, u) \in E\}$ denotes the set of vertices adjacent to $v$ (neighbors) and $d_v = |\Gamma_v|$ is the degree of $v$. $\Delta$ denotes the maximum degree of a vertex in the graph. In addition, $\Gamma_v^2 = \{w | (w, u) \in E \wedge w \neq v, \forall u \in \Gamma_v\}$ is the set of vertices that are exactly in distance 2 from $v$, i.e., neighbors of the neighbors of $v$ (excluding $v$ itself). A wedge in $G$ is a path of length two.

A biclique is a complete bipartite subgraph, and is parameterized by the number of vertices in each partition; for integers $\alpha, \beta$, an $\alpha \times \beta$ biclique in a bipartite graph is a complete subgraph with $\alpha$ vertices in $L$ and $\beta$ vertices in $R$. A butterfly in $G$ is a $2 \times 2$ biclique and consists of four vertices $(a, b, x, y) \in V$ where $a, b \in L$ and $x, y \in R$ such that edges $(a, x), (a, y), (b, x)$, and $(b, y)$ all exist in $E$. Let $\Sigma(G)$ denote the number of butterflies in $G$ (we use notation $\Sigma$ when $G$ is clear from the context). For vertex $v \in V$, let $\Sigma_v$ denote the number of butterflies that contain $v$. Similarly, for edge $e \in E$, let $\Sigma_e$ denote the number of butterflies containing $e$ (See Figure 1).

Our goal is to compute $\Sigma(G)$ for a graph $G$. We summarize our notation in Table 1.

When estimating the number of butterflies, we look for provable guarantees on the estimates computed using the following notion of randomized approximation. For parameters $\epsilon, \delta \in [0, 1]$, an $(\epsilon, \delta)$-approximation of a number $Z$ is a random variable $\tilde{Z}$ such that $\Pr[|\tilde{Z} - Z| > \epsilon Z] \leq \delta$.

Networks and Experimental Setup. Since we frequently present evaluation results close to the algorithm descriptions, we give some details about the data used for evaluation. We used massive real-world bipartite networks, selected from the publicly available KONECT network repository 1. The graphs that we have used are summarized in Table 2. We converted all graphs to simple, undirected graphs by removing edge directions (if graph was directed) and by removing multiple edges and vertices with degree zero.

![Figure 1: There are 4 butterflies in the entire graph, and the number of per-vertex and per-edge butterflies are shown for some vertices/edges.](image-url)

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$G = (V, E)$</td>
<td>simple bipartite graph with vertices $V$ and edges $E$</td>
</tr>
<tr>
<td>( V = L \cup R )</td>
<td>vertex partitions $L$ and $R$</td>
</tr>
<tr>
<td>$\Gamma_v$</td>
<td>set of vertices adjacent to $v$, i.e., ${u</td>
</tr>
<tr>
<td>$d_v$</td>
<td>degree of vertex $v$, i.e., $</td>
</tr>
<tr>
<td>$n, m, \wedge$</td>
<td>number of vertices ($</td>
</tr>
<tr>
<td>$\Delta$</td>
<td>maximum degree of a vertex in the graph</td>
</tr>
<tr>
<td>$\Gamma_v^2$</td>
<td>distance-2 neighbors of $v$ (excluding itself)</td>
</tr>
<tr>
<td>$\Sigma(G)$ or $\Sigma$</td>
<td>number of butterflies in graph $G$</td>
</tr>
<tr>
<td>$\Sigma_e$</td>
<td>number of butterflies that contain vertex $v$ (edge $e$)</td>
</tr>
</tbody>
</table>

1 ![http://konect.uni-koblenz.de/]
We implemented all algorithms in C++ and are publicly available on Github\(^2\) [24]. The source codes are compiled with Visual C++ 2015 compiler (Version 14.0), and report the runtimes on a machine equipped with a 3.50 GHz Intel(R) Xeon(R) CPU E3-1241 v3 and 16.0 GB memory.

3 RELATED WORK

**Bipartite graph motifs:** Modeling the smallest unit of cohesion enables a principled way to analyze networks. While the literature is quite rich with the studies on counting triangles and small cliques in unipartite graphs [3, 7, 11, 12, 21, 22, 27, 28], these works are not applicable to bipartite networks. To the best of our knowledge, Borgatti and Everett [6] are the first to consider cohesive structures in bipartite networks to analyze social networks. They proposed to use the 3 × 3 biclique as the smallest cohesive structure, motivated by the fact that a triangle in a unipartite graph has three vertices, and the same should be considered for both vertex sets of the bipartite graph. Opsahl also proposed a similar approach to define the clustering coefficient in affiliation networks [19]. Robins and Alexander argued that the smallest structure with multiple vertices on both vertex sets is a better model for measuring cohesion in bipartite networks [23], as also discussed in a later work [15]. They used the ratio of the number of 2 × 2 bicliques (butterflies) to the number of 3-paths (a path of three edges) to define the clustering coefficient in bipartite graphs. The butterfly is also adopted in a recent work by Axson et al. [4] to generate bipartite graphs with community structure. Butterfly counting is also applicable to the study of graphical codes (Haldor and Chugg [9]).

**Random Sampling for Motif Counting:** There has been much interest in recent years for counting motifs via random sampling, mostly focused on unipartite graphs and triangles. Works on triangle counting include edge sampling [29], subsequently improved by colorful sampling [20], on vertex and edge sampling [10, 32], wedge sampling [27], a hybrid scheme that considers the edge and wedge sampling [30], neighborhood sampling [21], and a recent space-efficient algorithm [8] based on reservoir-sampling.

### Table 2: Bipartite network datasets. \(L\) and \(R\) are vertex partitions, \(E\) is the edge set. The sum of degree squares for \(L\) and \(R\), and the number of butterflies are shown. \(K, M, B, T\) stand for \(10^3, 10^6, 10^9, 10^{12}\), respectively.

| Bipartite graph | \(|L|\) | \(|R|\) | \(|E|\) | \(\sum_{v \in L} d_v^2\) | \(\sum_{v \in R} d_v^2\) | \(\Xi\) |
|-----------------|-------|-------|-------|----------------|----------------|-------|
| DBpedia-Location | 172K  | 53K   | 293K  | 629K           | 245M           | 3.7M  |
| Wiki-fr         | 288K  | 3.9M  | 22M   | 1M             | 1B             | 2.2B  |
| Twitter         | 175K  | 530K  | 1.8M  | 74.1M          | 1.9B           | 206M  |
| Amazon          | 2.1M  | 1.2M  | 5.7M  | 82.5M          | 437M           | 35.8M |
| Journal         | 3.2M  | 7.4M  | 112M  | 9.5B           | 5.4T           | 3.3T  |
| Wiki-en         | 3.8M  | 21.4M | 122M  | 12.5T          | 23.2B          | 2T    |
| Deli            | 833K  | 53.7M | 101M  | 85.9B          | 52.7B          | 56.8B |
| Orkut           | 2.7M  | 8.7M  | 327M  | 156M           | 4.9T           | 22.1T |
| Web             | 27.6M | 12.7M | 140M  | 1.7T           | 211T           | 20T   |

\(^2\)https://github.com/beginner1010/butterfly-counting

---

### Algorithm 1: ExactBFC (V, E): Exact Butterfly Counting

**Input:** Graph \(G = (V = (L, R), E)\)

**Output:** \(\Xi(G)\)

1. \(\mathcal{A} \leftarrow \emptyset, \Xi \leftarrow 0\)
2. If \(\sum_{w \in L} (d_w)^2 < \sum_{v \in R} (d_v)^2\) then
3. \(\mathcal{A} \leftarrow R\)
4. For \(v \in \mathcal{A}\) do
5. \(C \leftarrow \text{hashmap}\) \(//\) initialized with zero
6. For \(u \in \Gamma_v\) do
7. \(\text{for } w \in \Gamma_u: w < v\) do
8. \(\Xi \leftarrow \Xi + |C[w]| + 1\) \(//\) dist-2 multiplicities
9. Return \(\Xi / 2\)

---

4 EXACT BUTTERFLY COUNTING

We first present the basic equation for the number of butterflies in a bipartite graph \(G\) and the base (state-of-the-art) algorithm by Wang et al. [31] that implements the equation.

**Lemma 1.** For a bipartite graph \(G = (V = (L \cup R), E)\),

1. \[\Xi(G) = \frac{1}{2} \sum_{v \in L} \Xi(v) - \frac{1}{2} \sum_{u \in L} \Xi(u).\]
2. \[\Xi(v) = \sum_{w \in \Gamma_v} \sum_{u \in \Gamma_u} \left| \Gamma_u \cap \Gamma_w \right| = \sum_{w \in \Gamma_v} \left| \Gamma_v \cap \Gamma_w \right|.
3. \[\Xi(G) = \frac{1}{2} \sum_{v \in L} \sum_{w \in \Gamma_v} \left| \Gamma_v \cap \Gamma_w \right|.

**Proof.** As each butterfly has exactly two vertices in the set \(L\), Equation (1) holds. For a vertex \(v \in L\), each butterfly it participates has one other vertex \(w \in L \setminus \{v\}\) and two vertices \(u, x \in R\). By definition, \(w \in \Gamma_v\). In order to find the number of pairs \(u, x\) that form a butterfly, we compute the intersection of the neighbor sets of \(v\) and \(w\). Number of the pairs in the intersection is defined by \(\left| \Gamma_v \cap \Gamma_w \right|\), as in Equation (2). Replacing \(\Xi(v)\) in Equation (1) by the right-most hand side of Equation (2), we obtain Equation (3).

An efficient implementation of Equation (3) is given in Algorithm 1 ExactBFC (ignore the colored lines). Instead of performing set intersection at each step, we count and store the number of paths from a vertex \(v \in L\) to each of its distance-2 neighbors \(w \in L\) by using a hash map \(C\) (in lines 5 to 8). Additional space complexity of ExactBFC is \(O(|V|)\), which is used by the hash map \(C\). Computational complexity of ExactBFC (without colored lines) is \(O(\sum_{w \in R} d_w)\).

**Lemma 2.** On input graph \(G = (V = (L \cup R), E)\), time complexity of ExactBFC (without colored lines) is \(O(\sum_{w \in R} d_w)\).
We compare the runtime of our algorithm (ExactBFC) with Wang et al. [31] (WFC). From our theoretical analysis in Section 4, our algorithm is expected to be faster than WFC. Figure 2 shows a comparison of the runtimes of the two algorithms. We note the following points. (1) ExactBFC is always faster than WFC. This also shows that the \( O(n) \) pre-processing step in ExactBFC to choose which side to proceed from is effective. (2) In many cases, ExactBFC achieves significant speedup when compared with WFC. This is especially true in cases where \( \sum_{e \in E} d_e^2 \) and \( \sum_{r \in R} d_r^2 \) differ significantly. In particular, ExactBFC is 700 times faster than WFC on Journal network and 35 times faster on Orkut. For the Web network, WFC did not complete in 40,000 seconds while ExactBFC completed in \( \approx 9,000 \) secs.

4.2 Local Butterfly Counting

We present two algorithms for local butterfly counting. \( \nuBFC \) (Algorithm 2) for counting the number of butterflies \( \nuBFC \), that contain a given vertex \( v \), and \( \epsilonBFC \) (Algorithm 3) for counting the number of butterflies \( \epsilon \), that contain an edge \( e \). Both algorithms employ procedures similar to the inner loop (lines 5 to 10) of ExactBFC. We will use \( \nuBFC \) and \( \epsilonBFC \) as building blocks in our sampling algorithms (Section 5).

Lemma 3. Time complexity for \( \nuBFC(v, G) \) is \( O(\Delta d_v) \) when \( \Delta \) is the maximum degree in \( G \), and for \( \epsilonBFC(e=(u, v), G) \) is \( O(|\Gamma_u^2| + |\Gamma_v|) = O(\Delta d_u + d_v) \) where \( d_u < d_v \), \( \text{w.l.o.g.} \).

Proof. \( \nuBFC \) iterates through each vertex in \( \Gamma_u^2 \) spending \( O(1) \) time per iteration. Since \( |\Gamma_u^2| \) is bounded by \( \Delta d_u \), the time complexity follows. For \( \epsilonBFC \), the only extra work is to check whether an \( x \in \Gamma_u^2 \) also exists in \( \Gamma_v \) and it can be done by a hash map which requires \( d_v \) preprocessing time.

5 APPROXIMATION BY LOCAL SAMPLING

In this section, we present approaches to approximating \( \nu(G) \) using random sampling. The intuition behind sampling is to examine a randomly sampled subgraph of \( G \) and compute the number of butterflies in the subgraph to derive an estimate of \( \nu(G) \). Since the subgraph is typically much smaller than \( G \), it is less expensive to perform an exact computation. The size of the chosen subgraph, the cost of computing on it, and the accuracy of the estimate vary according to the method by which we choose the random subgraph. This sampling process can be repeated multiple times, and averaged, in order to get a better accuracy.

We consider three natural sampling methods: vertex sampling (VSAMP), edge sampling (ESAMP), and wedge sampling (WSAMP). In VSAMP, the subgraph is chosen by first choosing a vertex uniformly at random, followed by the induced subgraph on the distance-2 neighborhood of the vertex. In ESAMP, a random edge is chosen, followed by the induced subgraph on the union of the immediate neighborhoods of the two endpoints of the edge. In WSAMP, a random wedge (path of length two) is chosen, followed by the induced subgraph on the intersection of the immediate neighborhoods of the two endpoints of the wedge. While the methods themselves are simple, the analysis of their accuracy involves having to deal carefully with the interactions of different butterflies being sampled together.

5.1 Vertex Sampling (Algorithm VSAMP)

The idea in VSAMP is to sample a random vertex \( v \) and count the number of butterflies that contain \( v \). This is accomplished by counting the number of butterflies in the induced subgraph consisting of
Let \( G \) be a graph, \( v \) be a vertex, and \( X \) be the set of all \( n \) butterflies of that type. In addition, let \( p_{\text{v}} \) denote the number of pairs of butterflies of that type. In order to determine \( X \), we consider the joint probabilities of \( V \) being sampled together. The set of all pairs of butterflies is partitioned into different types as follows. This partitioning will help not only in analyzing this algorithm, but also in subsequent sampling algorithms. A pair of butterflies is said to be of type:

- 0v: if they share zero vertices (Figure 3a)
- 1v: if they share one vertex (Figure 3b)
- 2v: if they share two vertices but no edge (Figure 3c)
- 1e: if they share one edge in common but no edge (Figure 3d)
- 1w: if they share three vertices and exactly one edge (Figure 3d)
- 1w: if they share three vertices and two edges i.e. share a wedge (Figure 3e)

It can be verified that every pair of distinct butterflies must be one of the above types \( \{0v, 1v, 2v, 1e, 1w\} \), and other combinations such as three vertices and more than two edges are not possible. For each type \( t \in \{0v, 1v, 2v, 1e, 1w\} \), let \( p_t \) denote the number of pairs of butterflies of that type. In addition, let \( p_{\text{v}} = p_{1v} + p_{2v} + p_{1e} + p_{1w} \) be the number of pairs of butterflies that share at least one vertex.

\[
\text{Var}[Y_v] = \text{Var} \left[ \frac{X}{4} \sum_{t=1}^{16} \text{Var}[X_t] \right] = \frac{n^2}{16} \sum_{t=1}^{16} \text{Var}[X_t] + \sum_{t \neq j} \text{Cov}[X_t, X_j] = \frac{n^2}{16} \sum_{t=1}^{16} \text{Var}[X_t] + \sum_{t \neq j} \text{Cov}[X_t, X_j]
\]

Consider the different types of butterfly pairs \( (i, j) \):

- Type 0v, there is zero probability of \( i \) and \( j \) being counted within \( X \), hence \( \text{Var}[X_i, X_j] = 0 \). \( \text{Cov}(X_i, X_j) = 16/n^2 \).
- Type 1v, \( \text{E}[X_i, X_j] = \text{Pr}[X_i = 1] \text{Pr}[X_j = 1] = 1/n \). \( \text{Cov}(X_i, X_j) = 1/n - 16/n^2 \).
- Type 2v, \( \text{E}[X_i, X_j] = (4/n)(1/2) = 2/n \). \( \text{Cov}(X_i, X_j) = 2/n - 16/n^2 \).
- Type 1e, \( \text{E}[X_i, X_j] = (4/n)(1/2) = 2/n \). \( \text{Cov}(X_i, X_j) = 2/n - 16/n^2 \).
- Type 1w, \( \text{E}[X_i, X_j] = (4/n)(3/4) = 3/n \). \( \text{Cov}(X_i, X_j) = 3/n - 16/n^2 \).

By ignoring the negative contributions and adding up the other contributions, we arrive:

\[
\text{Var}[Y_v] \leq \frac{n}{4} \text{Var} + \frac{n}{8} p_{1v} + \frac{n}{16} p_{2v} + \frac{n}{16} p_{1e} + \frac{n^3}{16} p_{1w} \leq \frac{n}{4} \text{Var} + p_{\text{v}}
\]
Algorithm 5: ESAMP (single iteration)

Input: A bipartite graph \( G = (V, E) \)
Output: An estimate of \( \Xi(G) \)
1. Choose an edge \( e \) from \( E \) uniformly at random.
2. \( Z \leftarrow eBFC(e, G) \) // Algorithm 3
3. return \( \Xi(e) \cdot m/4 \)

Let \( Z \) be the average of \( a = \frac{2a_2}{\Xi} \left( 1 + \frac{p_Y}{\Xi} \right) \) independent instances of \( Y \). Using \( \var{\alpha} = \var{Y_1} / \alpha \) and Chebyshev’s inequality:
\[
\Pr\left[ \left| Z - \Xi \right| \geq \epsilon \Xi \right] \leq \frac{\var{Y_1}}{\epsilon^2 \Xi^2} \leq \frac{\var{Y_1} + p_Y}{\epsilon^2 \Xi^2} \leq \frac{n(\Xi + p_Y)}{44\epsilon^2 \Xi^2} = \frac{1}{32}
\]

We can turn the above estimator into an \((\epsilon, \delta)\) estimator by taking the median of \( O(\log(1/\delta)) \) estimators, using standard methods.

Lemma 5. There is an algorithm that uses \( \tilde{O} \left( \frac{n}{\epsilon} \left( 1 + \frac{p_Y}{\Xi} \right) \right) \) iterations \(^4\) of VSAMP (Algorithm 4) and yields an \((\epsilon, \delta)\)-estimator of \( \Xi(G) \) using expected time \( \tilde{O} \left( \frac{n^2}{\epsilon^2} \left( 1 + \frac{p_Y}{\Xi} \right) \right) \). Expected additional space is \( O \left( \frac{m \Delta}{\epsilon \Xi} \right) \).

Proof. An iteration of VSAMP samples a vertex \( v \) and calls VBFC (Algorithm 2) for local butterfly counting once, which takes \( O(|\Gamma_v^0|) \) time. Hence, the expected runtime of an iteration is \( \tilde{O} \left( \frac{n}{\epsilon} \left( 1 + \frac{p_Y}{\Xi} \right) \right) \) time, where the expectation is taken over a uniform random choice of a vertex. We note that \( |\Gamma_v^0| \leq d_v \Delta \) where \( d_v \) is \( \Delta \)‘s degree and \( \Delta \) is the maximum degree in the graph. Thus \( \mathbb{E}[|\Gamma_v^0|] \leq \sum_{v \in V} \frac{1}{\Xi} d_v \Delta \leq \frac{\Delta}{\Xi} \sum_{v \in V} d_v = \frac{m \Delta}{\Xi} \). The space of VSAMP is same with VBFC (Algorithm 2); \( O(|\Gamma_v^0|) \) for handling vertex \( v \). The expected value is \( \tilde{O} \left( \frac{m \Delta}{\epsilon \Xi} \right) \).

5.2 Edge Sampling (Algorithm ESAMP)

In this algorithm, the idea is to sample a random edge and count the number of butterflies that contain this edge, using eBFC (Algorithm 3). We present ESAMP in Algorithm 5, and state its properties. The proofs of the lemmas are omitted due to space constraints, and can be found in the full version [25].

Let \( Y_E \) denote the return value of ESAMP (Algorithm 5). Let \( p_E \) be the number of pairs of butterflies that share at least one edge. Then, \( Y_E = p_{te} + p_{tw} \).

Lemma 6. \( \mathbb{E}[Y_E] = \Xi \) and \( \mathbb{V}[Y_E] \leq \frac{\var{p}}{\Xi} + p_E \). Using \( \tilde{O} \left( \frac{\alpha^2}{\epsilon^2} \left( 1 + \frac{p_Y}{\Xi} \right) \right) \) iterations of ESAMP yields an \((\epsilon, \delta)\)-estimator of \( \Xi(G) \) using time \( \tilde{O} \left( \frac{m \Delta}{\Xi} \left( 1 + \frac{p_Y}{\Xi} \right) \right) \). The additional space complexity is \( O \left( \frac{m \Delta}{\epsilon \Xi} \right) \).

5.3 Wedge Sampling (Algorithm WSAMP)

In WSAMP, we first choose a random ‘wedge’, a path of length two in the graph. This already yields three vertices that can belong to a potential butterfly. Then, we count the number of butterflies that contain this wedge by finding the intersection of the neighborhoods of the two endpoints of the wedge. Algorithm 6 describes the WSAMP algorithm. The correctness and complexity are stated

\(^4\)We use the notation \( \tilde{O}(f) \) to suppress the factor \( \frac{\log(1/\delta)}{\epsilon^2} \), i.e. mean \( O \left( \frac{\log(1/\delta)}{\epsilon^2} \right) \).

Algorithm 6: WSAMP (single iteration)

Input: A bipartite graph \( G = (V, E) \)
Output: An estimate of \( \Xi(G) \)
1. \( \Xi \leftarrow \sum_{u \in V} \left( \frac{d_u}{2} \right) \) // number of wedges in \( G \)
2. Choose a vertex \( u \in V \) with probability \( \left( \frac{d_u}{\Xi} \right) / \Xi \)
3. Choose two distinct vertices \( u, w \in \Gamma_u \) uniformly at random
4. \( \beta \leftarrow |\Gamma_u \cap \Gamma_w| \) // # butterflies that has \((u, v, w)\)
5. return \( \beta \cdot \Delta / 4 \)

Table 3: Standard deviations of estimators on large graphs.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>( \sqrt{\mathbb{V} p_E} ) (m)</th>
<th>error %</th>
<th>( \sqrt{\mathbb{V} p_{tw}} ) (m)</th>
<th>error %</th>
<th>( \sqrt{\mathbb{V} p_{te}} ) (m)</th>
<th>error %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deli</td>
<td>2.8 x 10^{-10}</td>
<td>194.9</td>
<td>2.1 x 10^{-12}</td>
<td>38.3</td>
<td>7.7 x 10^{-11}</td>
<td>13.6</td>
</tr>
<tr>
<td>Journal</td>
<td>2.3 x 10^{-10}</td>
<td>708.56</td>
<td>2.1 x 10^{-13}</td>
<td>6.4</td>
<td>1.4 x 10^{-14}</td>
<td>99.3</td>
</tr>
<tr>
<td>Orkut</td>
<td>5 x 10^{-10}</td>
<td>223.6</td>
<td>2 x 10^{-13}</td>
<td>9.2</td>
<td>2.9 x 10^{-13}</td>
<td>13.3</td>
</tr>
<tr>
<td>Web</td>
<td>6.8 x 10^{-10}</td>
<td>3431.5</td>
<td>8.2 x 10^{-14}</td>
<td>4.1</td>
<td>4.3 x 10^{-14}</td>
<td>2194.8</td>
</tr>
<tr>
<td>Wiki-en</td>
<td>1.3 x 10^{-10}</td>
<td>623.6</td>
<td>3.8 x 10^{-13}</td>
<td>19</td>
<td>1.4 x 10^{-13}</td>
<td>703.4</td>
</tr>
</tbody>
</table>

Table 3 shows the variances of \( \Xi(G) \). The results show that the variances of \( \sqrt{\mathbb{V} p_{te}} \), \( \sqrt{\mathbb{V} p_{tw}} \), and \( \sqrt{\mathbb{V} p_E} \) are comparable to the number of edges \( E \), while \( \sqrt{\mathbb{V} p_{te}} \) is much smaller than \( E \) in most of these graphs, but \( \sqrt{\mathbb{V} p_{tw}} \) is much smaller than \( E \).

5.4 Accuracy and Runtime of Sampling

Accuracy of a Single Iteration. In order to understand the relation between the three sampling algorithms, we compare the variance of the estimates returned by these algorithms. Note that each of them returns an unbiased estimate of the number of butterflies. The standard deviations (square root of variances) of VSAMP, ESAMP, and WSAMP for different graphs are estimated and summarized in Table 3. The results show that the variances of VSAMP and WSAMP are much higher than the variance of ESAMP. Note that these are estimates of an upper bound on the variances, and the actual variances could be (much) smaller.

The variance of VSAMP is proportional to \( np_Y \) where \( p_Y \) is the number of pairs of butterflies that share a vertex and \( n \) is the number of vertices. The variance of ESAMP is proportional to \( mp_E \) where \( p_E \) is the number of pairs of butterflies that share an edge and \( m \) is the number of edges. Note that typically \( \Xi \propto p_Y \) and \( \Xi + p_Y \propto p_Y \). If two butterflies share an edge, they share a vertex, hence \( p_E \leq p_Y \). Since it is possible that two butterflies share a vertex but do not share an edge, \( p_E \) could be much smaller than \( p_Y \). It turns out that in most of these graphs, \( p_E \) is much smaller than \( p_Y \). On the other hand, the number of vertices in a graph \( (n) \) is comparable to the number of edges \( (m) \). Typically \( m \approx 10n \), and only in one case (Orkut), we have \( m \approx 30n \). Thus, \( np_Y \propto mp_E \) for the graphs we consider. As a result, the variance of VSAMP is much larger than the variance of ESAMP, which is reflected clearly in Table 3.
Algorithm 7: Fast-eBFC (replaces eBFC in ESAMP)

**Input:** An edge $e = (u, v) \in E$ in $G = (V, E)$

**Output:** An estimate of $\Sigma_e$

1. Choose a vertex $w$ from $\Gamma_u$ uniformly at random
2. Choose a vertex $x$ from $\Gamma_v$ uniformly at random
3. If $(u, v, w, x)$ forms a butterfly then $\beta \leftarrow 1$
4. Else $\beta \leftarrow 0$
5. Return $\beta \cdot d_u \cdot d_v$

Comparing ESAMP with WSAMP, we note that the variance of WSAMP is proportional to $\Lambda \cdot p_{1w}$, where $\Lambda$ is the number of wedges in the graph ($O(\Sigma_e d^2_e)$) and $p_{1w}$ is the number of pairs of butterflies that share a wedge. The variance of ESAMP is proportional to $m \cdot p_E$, $p_{1w} \leq p_E$ since each pair of butterflies that shares a wedge also shares an edge. At the same time, we see that $\Lambda$ is substantially greater than $m$. Overall, there is no clear winner among WSAMP and ESAMP in theory, but ESAMP seems to have the smaller variance on real-world networks, typically, sometimes much smaller, as in graph Wiki-en.

**Runtime per Iteration:** The performance of a sampling algorithm depends not only on the variance of an estimator, but also on how quickly an estimator can be computed. Figure 4 shows the time taken to compute a single estimator using different sampling algorithms for the five largest networks. We note that ESAMP (which calls eBFC) requires the largest amount of time per sampling step, among all estimators. This decreases the overall accuracy of ESAMP, despite its smaller variance.

5.5 Faster Edge Sampling using Fast-eBFC

Since ESAMP had a low variance, but a high runtime per iteration, we tried to achieve different tradeoffs with respect to runtimes and accuracy, which led to our next algorithm Fast-eBFC, a faster variant of butterfly counting per edge (eBFC). Like ESAMP, we first sample a random edge from the graph, but instead of exactly counting the number of butterflies that contain the sampled edge, which leads to (relatively) expensive iterations, Fast-eBFC only estimates the number of butterflies per edge, through a further sampling step (replacing eBFC in line 2 of ESAMP (Algorithm 5)). For an edge $(u, v)$ this estimation is performed by randomly choosing one neighbor each of $u$ and of $v$, and checking if the four vertices form a butterfly. Algorithm 7 presents a single iteration of Fast-eBFC. This procedure is repeated a few times for a given edge, to improve the accuracy of the estimate. In our implementation we repeated it 1000 times for each sampled edge, and it was still significantly faster than eBFC (Figure 4). We use it instead of the eBFC algorithm in ESAMP. While the estimate from each iteration is less accurate than in ESAMP, more iterations are possible within the same time. Fast-eBFC (with 1000 repetitions) is faster than eBFC by 15x-357x, when used in ESAMP. Overall, in large graphs, this leads to an improvement in accuracy over eBFC in ESAMP. Let $Y_{FE}$ denote the return value of Algorithm 7.

Let $Z$ be the average of $\alpha = 32(d_u \cdot d_v)/\varepsilon^2 \Sigma_e$ independent instances of $Y_{FE}$. Using Chebyshev’s inequality, we arrive that $Z$ is an $(\varepsilon, 1/32)$-estimator of $\Sigma_e$.

**Fast Wedge Sampling:** We also experimented with a faster version of WSAMP, where the number of butterflies containing a wedge was estimated using sampling. But this did not yield good results, partly because the time for each iteration of WSAMP was already relatively small. For example, in Dei1 graph, WSAMP had less than $2\%$ error after 5 secs while Fast Wedge Sampling ended up with $29\%$ error after 10 secs. In web also the error percentage of Fast Wedge Sampling was $60\%$ higher than WSAMP’s.

5.6 Comparing Sampling-based Approaches

We compare the sampling algorithms VSAMP, ESAMP, eBFC, ESAMP+Fast-eBFC, and WSAMP. A sampling algorithm immediately starts producing estimates that get better as more iterations are executed. We record the number of iterations and relative percent error of sampling algorithms up to 60 seconds. Figure 5 shows the relative percent error with respect to the runtime for five large bipartite graphs. We report the median error of the 30 trials of sampling methods for each data point.

Our experiments show that VSAMP performs poorly when compared with ESAMP and WSAMP under the same time budget – this is along expected lines, based on our analysis of the variance. The accuracy of ESAMP and WSAMP are comparable, with ESAMP being slightly better. Note that ESAMP has a better variance, while WSAMP has faster iterations. For instance, on the Wiki-en network, 1000 iterations of WSAMP yields $51\%$ error in 0.13 secs, whereas 1000 iterations of ESAMP yields $29\%$ error in 33 seconds. Across all the graphs, ESAMP using Fast-eBFC, which combines the benefits of faster iterations with a good variance, yields the best results, and is superior to all other sampling methods; it leads to less than $1\%$ relative error within 5 seconds.

6 APPROXIMATION BY ONE-SHOT SPARSIFICATION

We present methods for estimating $\Sigma(G)$ using one-shot sparsification – where we thin down the input graph into a smaller graph through a global sampling step. The number of butterflies in the sparsified graph is used to estimate $\Sigma(G)$. Unlike algorithms such as ESAMP and WSAMP, which work on a small subgraph constructed around a single randomly sampled edge or a wedge, sparsification methods put each edge in the graph into the sample with a certain probability. We consider two approaches to sparsification: (1) ESPAR in Section 6.1, where edges are chosen independently of each other and (2) ClRSpar in Section 6.2, based on a sampling method where different edges are not independently chosen, but dense regions appear with higher probability in the sample – based on a similar idea in the context of triangle counting due to Pagh and Tsourakakis [20].

6.1 Edge Sparsification (ESPAR)

In ESPAR, the input graph $G$ is sparsified by independently retaining each edge in $G$ in the sampled graph $G'$, with a probability $p$. The number of butterflies in $G'$, obtained using ExactBFC, is used to construct an estimate of $\Sigma(G)$, after applying a scaling factor. This
has to deal with the fact that although different edges are sampled
found in the full version of the paper [25]. The proof of the expec-
tation is straightforward. For bounding the variance, the analysis
has to deal with the fact that although different edges are sampled
independently, the random variables corresponding to different
butterflies being sampled are not independent of each other, since
butterflies may share edges. By accounting for the covariances, we
arrive at a bound on the variance of $Y_{ES}$ using the following observation (whose proof is in [25]), which allows a bound on the variance in terms of $\Delta$ and $\Delta^c$.

**Observation 1.** $p_{2k} \leq \Delta^2, p_{1e} \leq \Delta^2, and p_{1w} \leq \Delta^2$.

Using Chebyshev’s inequality and standard methods, the estimator $Y_{ES}$ can be repeated $O((1/\epsilon)^2)$ times to get an $(\epsilon, \delta)$ estimator of $\Delta(G)$.

6.2 **Colorful Sparsification (Clr Spar)**

The idea in Clr Spar is to sample edges at a rate of $p$, as in E Spar, but add dependencies between the sampling of different edges, such that there is a greater likelihood that a dense structure, such as the butterfly, is preserved in the sampled graph. The algorithm randomly assigns one of $N$ colors to vertices, and sample only those
edges whose endpoints have the same color. The ClrSpar algorithm for approximate butterfly counting is presented in Algorithm 9.

We developed this method due to the following reason. Suppose $p = 1/N$. Though the expected number of edges in the sampled graph is $mp$, the same as in ESpar, it can be seen that the expected number of butterflies in the sampled graph is equal to $p^3 \Sigma$, which is higher than in the case of ESpar ($p^3 \Sigma$). Thus, for a sampled graph of roughly the same size, we expect to find more butterflies in the sampled graph. Note however that this does not directly imply a lower variance of the estimator due to ClrSpar.

**Lemma 10.** Let $Y$ be the output of ClrSpar on input $G$, and $p = 1/N$. $\mathbb{E}[Y] = \Sigma(G)$. If $p > \max(\sqrt{3}/\sqrt{2}, \sqrt{\frac{\Delta}{2N}})$, then $\text{Var}[Y] \leq \Sigma^2/8$.

### 6.3 Comparison of Sparsification Algorithms

The parameter $p$ controls the probability of an edge being included in the sparsified graph. As $p$ increases, we expect the accuracy as well as the runtime to increase. The relative accuracies of the two methods depend on the variances of the estimators. We estimated the variances using our analysis, and Table 4 presents the results. We observe that the variance of ESpar is predicted to be much lower than that of ClrSpar. To understand this, we begin with Lemmas 9 and 10 which show expressions bounding the variances in terms of $p_{1w}, p_{1e}$, and $p_{2v}$, also summarized in Table 4. The difference in the variance between ClrSpar and ESpar boils down to $(\Sigma p^{-3} + p_{2v} p^{-1} - \Sigma p^{-4})$. In our experiments, we found that $p_{2v}$, the number of pairs of butterflies that share two vertices without sharing an edge, was very high, much larger than $p_{1e}, p_{1w}$, and $\Sigma$. This explains why the variance of ClrSpar was higher.

This observation about variance is consistent with our experimental results. In Figure 6, we report the relative percent error as well as the runtime as the sampling probability $p$ increases. Results show that ESpar obtains less than one percent error when 5 percent of edges are sampled. However, as shown in Figures 6b, 6d and 6e, ClrSpar requires a larger sampling probability to achieve a reasonable accuracy.

<table>
<thead>
<tr>
<th>Graph</th>
<th>ESAMP with Fast-eBFC</th>
<th>ESpar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deli</td>
<td>3.4</td>
<td>2.1</td>
</tr>
<tr>
<td>Journal</td>
<td>5.0</td>
<td>1.7</td>
</tr>
<tr>
<td>Orkut</td>
<td>3.4</td>
<td>3.4</td>
</tr>
<tr>
<td>Web</td>
<td>4.1</td>
<td>3.9</td>
</tr>
<tr>
<td>Wiki-en</td>
<td>4.8</td>
<td>2.3</td>
</tr>
</tbody>
</table>

**Table 5:** Time (in seconds) to obtain 1% relative percent error for the best sampling and sparsification algorithms.

### 6.4 Sampling or Sparsification?

The accuracy of the best sampling algorithm, ESAMP with Fast-eBFC, is compared with the best sparsification algorithm, ESpar, in Table 5. Overall, two algorithms take similar times to reach a 1% error on all graphs we considered, in the range of 1.7-5 sec, with ESpar achieving this accuracy faster than ESAMP with Fast-eBFC.

However, ESpar has other downsides when compared with ESAMP. First, the memory consumption of ESpar is $O(mp)$ where $p$ is a parameter, and is larger than ESAMP with Fast-eBFC, whose memory consumption is $O(\Delta)$. As a result, we expect the memory of ESpar to be linearly in the size of the graph, showing that it may be easier for ESAMP to scale to graphs of even larger sizes. Next, ESpar needs to decide on a sampling parameter $p$ to balance between accuracy and runtime. If $p$ is too large, then the runtime is high, and if $p$ is too small, then the accuracy is low. Finally, one-shot sparsification algorithms assume that the entire graph is available whereas the ESAMP needs access to only a subgraph of the graph. Thus if one has to pay for data about the graph, or if the data about edges is hard to obtain, then sampling may be the better alternative.

Overall, local sampling algorithms can find a larger application space in real world problems. If the entire graph is available, and memory is not a bottleneck, it may be better to go with ESpar. For more restrictive scenarios, ESAMP combined with Fast-eBFC is the better option.

### 7 CONCLUSION

We introduced a suite of algorithms for butterfly counting in bipartite networks. We first showed that a simple statistic about vertex sets, which is cheap to obtain, helps drastically to reduce the runtime of exact algorithms. We then presented scalable randomized algorithms that approximate the number of butterflies in a graph, with provable accuracy guarantees.

Randomized algorithms using one-shot sparsification are applicable when the entire graph is available locally, and rely on a global sampling step to compute a smaller "sparsified" subgraph, which is used to compute an accurate estimate. On the other hand, if the access to the graph data is limited, local sampling algorithms can be used to randomly sample small subgraphs of the entire graph, and analyze them to compute an estimate. Sampling algorithms are especially beneficial when there is a rate-limited API that provides random samples of the network data, such as the GNIP [1] and Facebook Graph API [2]. Our best sampling and sparsification algorithms yield less than 1% relative error within $\approx 4$ and $\approx 5$ seconds for all the networks we considered, whereas the state-of-the-art exact algorithm does not complete even in 40,000 secs on the Web graph.
There are many promising directions to follow. Here, we only focused on the butterfly motif, but one can consider other motifs suitable to bipartite graphs. Note that the combinatorial explosion for the number of butterflies is more serious than the triangles in unipartite graphs. Thus, it is challenging to consider even larger motifs, but the ideas in this work can serve as building blocks for considering different motifs. Another direction is adapting our algorithms for the streaming scenario. Given a single pass over the graph, the question is how to sample/sparsify the graph stream to accurately estimate the number of butterflies. Sparsification-based algorithms may be adapted to the streaming scenario quite easily, but same cannot be said for the local sampling algorithms.

ACKNOWLEDGMENT

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

REFERENCES