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:
Seyed-Vahid Sanei-Mehri, Ahmet Erdem Sarıyüce, and Srikanta Tirthapura. 2018. Butterfly Counting in Bipartite Networks. In *KDD '18: The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, August 19–23, 2018, London, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3219819.3220097

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, artificially 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.

KDD '18, August 19–23, 2018, London, United Kingdom

ACM ISBN 978-1-4503-5552-0/18/08…$15.00
https://doi.org/10.1145/3219819.3220097

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+FastBFC, gives estimates with a relative error less than 1 percent 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.

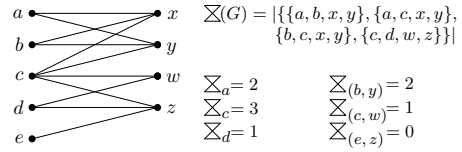


Figure 1: There are 4 butterflies in the entire graph, and the number of per-vertex and per-edge butterflies are shown for some vertices/edges.

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. Δ 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 α, β, an $\alpha \times \beta$ biclique in a bipartite graph is a complete subgraph with α vertices in L and β vertices in R. A butterfly in G is a 2×2 biclique and consists of 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) all exist in E. Let $\bowtie(G)$ denote the number of butterflies in G (we use notation $\bowtie$ when G is clear from the context). For vertex $v \in V$, let $\bowtie_v$ denote the number of butterflies that contain v. Similarly, for edge $e \in E$, let $\bowtie_e$ denote the number of butterflies containing e (See Figure 1). Our goal is to compute $\bowtie(G)$ 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 (ϵ, δ) – approximation of a number Z is a random variable $\hat{Z}$ such that $\Pr[|\hat{Z} - Z| > \epsilon Z] \leq \delta$.

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.

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

$G = (V, E)$	simple bipartite graph with vertices V and edges E
$V = L \cup R$	vertex partitions L and R
Γ_v	set of vertices adjacent to v, i.e., $\{u \mid (v, u) \in E\}$
d_v	degree of vertex v, i.e., $\lvert\Gamma_v\rvert$
$n, m, \wedge$	number of vertices ($\lvert V\rvert$), edges ($\lvert E\rvert$), and wedges ($\sum_{v \in V} \binom{d_v}{2}$)
Δ	maximum degree of a vertex in the graph
Γ_v^2	distance-2 neighbors of v (excluding itself)
$\bowtie(G)$ (or $\bowtie$)	number of butterflies in graph G
$\bowtie_{v(e)}$	number of butterflies that contain vertex v (edge e)

Table 1: Notations

Bipartite graph	$\|L\|$	$\|R\|$	$\|E\|$	$\sum_{\ell \in L} d_\ell^2$	$\sum_{r \in R} d_r^2$	$\bowtie$
$10^4 \times 10$ biclique	10K	10	100K	1M	1B	2.2B
DBPedia-Location	172K	53K	293K	629K	245M	3.7M
Wiki-fr	288K	3.9M	22M	2.1T	795M	601B
Twitter	175K	530K	1.8M	74.1M	1.9B	206M
Amazon	2.1M	1.2M	5.7M	828M	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	33.7M	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

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.

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 Aksoy et al. [4] to generate bipartite graphs with community structure. Butterfly counting is also applicable to the study of graphical codes (Halford and Chugg [9]). The numbers of cycles of length g, $g+2$, and $g+4$ in bipartite graphs, where g is the girth[3], characterize the decoding complexity – note that the butterfly is the simplest non-trivial cycle in a bipartite graph.

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. To the best of our knowledge, random sampling for butterfly counting in bipartite networks has not been studied in the past.

Butterfly Counting: The closest work to ours is by Wang et al. [31], who presented exact algorithms for butterfly (rectangle) counting. Their algorithms outperform generic matrix-multiplication based methods for counting cycles in a graph [5]. We present an improved algorithm for exact butterfly counting, and then present more efficient randomized algorithms for approximate butterfly counting.

[2] https://github.com/beginner1010/butterfly-counting
[3] length of the shortest cycle in a graph

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

Input : Graph $G = (V = (L, R), E)$
Output: $\bowtie(G)$

1 $\mathcal{A} \leftarrow L$, $\bowtie \leftarrow 0$
2 **if** $\sum_{u \in L} (d_u)^2 < \sum_{v \in R} (d_v)^2$ **then**
3 $\mathcal{A} \leftarrow R$
4 **for** $v \in \mathcal{A}$ **do**
5 $C \leftarrow hashmap$ // initialized with zero
6 **for** $u \in \Gamma_v$ **do**
7 **for** $w \in \Gamma_u : w \prec v$ **do**
8 $C[w] \leftarrow C[w] + 1$ // dist-2 multiplicities
9 **for** $w \in C : C[w] > 0$ **do**
10 $\bowtie \leftarrow \bowtie + \binom{C[w]}{2}$
11 **return** $\bowtie/2$ ($\bowtie$)

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) $\bowtie(G) = \frac{1}{2} \sum_{v \in L} \bowtie_v$.
(2) $\bowtie_v = \sum_{u \in \Gamma_v} \sum_{(\text{unique}) w \in \Gamma_u \setminus v} \binom{|\Gamma_v \cap \Gamma_w|}{2} = \sum_{w \in \Gamma_v^2} \binom{|\Gamma_v \cap \Gamma_w|}{2}$.
(3) $\bowtie(G) = \frac{1}{2} \sum_{v \in L} \sum_{w \in \Gamma_v^2} \binom{|\Gamma_v \cap \Gamma_w|}{2}$.

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 $w \in L$ ($w \neq v$) and two vertices $u, x \in R$. By definition, $w \in \Gamma_v^2$. In order to find the number of u, x pairs that v and w 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 $\binom{|\Gamma_v \cap \Gamma_w|}{2}$, as in Equation (2). Replacing $\bowtie_v$ 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 neighbor $w \in L$ 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 given by the following.

Lemma 2. *On input graph $G = (V = (L \cup R), E)$, time complexity of* ExactBFC *(without colored lines) is $O(\sum_{u \in R} d_u^2)$.*

Algorithm 2: vBFC (v, G): Per Vertex Butterfly Counting

Input: A vertex $v \in V$ in $G = (V = (L \cup R), E)$
Output: $\bowtie_v$, number of butterflies in G that contain v

1 $\bowtie_v \leftarrow 0$, $C \leftarrow$ *hashmap* // initialized with zero
2 **for** $u \in \Gamma_v$ **do**
3 **for** $w \in \Gamma_u$ **do if** $w \neq v$ **then** $C[w] \leftarrow C[w] + 1$
4 **for** $w \in C$ **do** $\bowtie_v \leftarrow \bowtie_v + \binom{C[w]}{2}$
5 **return** $\bowtie_v$

Proof. For each $v \in L$, we find a distance-2 neighbor vertex $w \in L$ such that there exists a $u \in R$ where $(v, u) \in E$ and $(w, u) \in E$. In lines 6 to 8, the nested `for` loop performs $O(1)$ computation for each tuple (v, u, w) where $v \in L, u \in \Gamma_v$, and $w \in \Gamma_u$. *The number of such triples is exactly the number of paths in the graph of length two, with the midpoint (u) in R*, which is equal to $\sum_{u \in R} \binom{d_u}{2} = O(\sum_{u \in R} (d_u)^2)$. □

We have two observations about Equation (3) and Algorithm 1. First, the intersection operation need not be performed for each ordered pair v, w in L, but instead can be performed for all (ordered) pairs $w \prec v$, thus preventing double counting of a butterfly.
Second, instead of iterating over the vertex set L in Equation (3), we can also use other vertex set, R;
(4) $\bowtie(G) = \frac{1}{2} \sum_{u \in R} \sum_{x \in \Gamma_u^2} \binom{|\Gamma_u \cap \Gamma_x|}{2}$
Clearly, Equations (3) and (4) give the same answer overall, but their computational costs can be significantly different. We show (Lemma 2) that the runtime of the algorithm is $O(\sum_{u \in R} (d_u)^2)$ if we use $\mathcal{A} = L$. Based on this analysis, we propose to use an $O(n)$ time pre-computation step that compares the sum of degrees in L and R to choose the cheaper option. If $\sum_{v \in L} (d_v)^2 < \sum_{u \in R} (d_u)^2$, then we choose the right side, R; otherwise we choose the left side, L. This is done in the lines 2 and 3 (in pink).

ExactBFC algorithm (including colored lines) has the complexity of $O(min(\sum_{u \in R} (d_u)^2, \sum_{v \in L} (d_v)^2)$ and improves upon the time complexity of the algorithm due to Wang et al. [31], which is $O\left(\sum_{u \in R} (d_u)^2\right)$. The main difference is that their algorithm always starts from the left vertex set L, while we choose the cheaper option depending on the sum of degrees in each side.

4.1 Performance of Exact Butterfly Counting

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 $\sum_{\ell \in L} d_\ell^2$ 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.

Algorithm 3: eBFC (e, G): Per Edge Butterfly Counting

Input: An edge $e = (u, v) \in E$ in $G = (V, E)$
Output: $\bowtie_e$, number of butterflies in G that contain e

1 $\bowtie_e \leftarrow 0$
2 **for** $w \in \Gamma_u \setminus \{v\}$ **do**
3 **for** $x \in \Gamma_w$ **do if** $x \in \Gamma_v \setminus \{u\}$ **then** $\bowtie_e \leftarrow \bowtie_e + 1$
4 **return** $\bowtie_e$

4.2 Local Butterfly Counting

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

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

Proof. vBFC iterates through each vertex in Γ_v^2 spending $O(1)$ time per iteration. Since $|\Gamma_v^2|$ is bounded by $\Delta\ d_v$, the time complexity follows. For eBFC, the only extra work is to check whether an $x \in \Gamma_u^2$ also exists in Γ_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 $\bowtie(G)$ 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 $\bowtie(G)$. 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

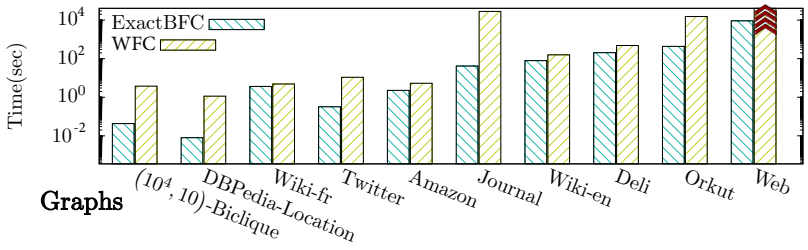
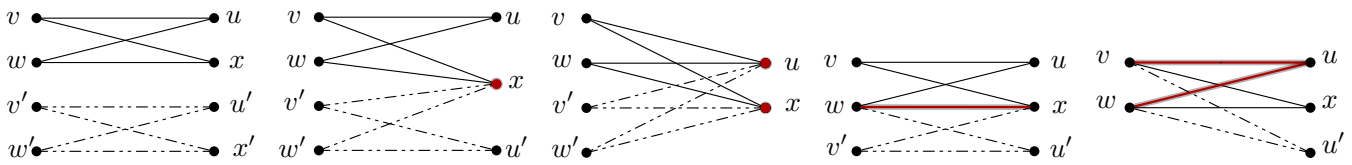


Figure 2: Runtimes for Exact Butterfly Counting, showing speedups up to 3 orders of magnitude for ExactBFC over WFC. For the Web graph, WFC did not finish in 40,000 secs.



(a) Type $0v$: No vertex in common **(b) Type $1v$: One vertex in common** **(c) Type $2v$: Two vertices in common but no edge** **(d) Type $1e$: One edge in common** **(e) Type $1w$: One wedge in common**

Figure 3: A pair of butterflies in G can be of one of the above five types.

Algorithm 4: VSamp (single iteration)

Input: A bipartite graph $G = (V, E)$
Output: An estimate of $\bowtie(G)$

1 Choose a vertex v from V uniformly at random.
2 $\bowtie_v \leftarrow$ vBFC (v, G) `// Algorithm 2`
3 **return** $\bowtie_v \cdot n/4$

the distance-2 neighborhood of v in the graph. We show that the algorithm, described in Algorithm 4, yields an unbiased estimate of $\bowtie(G)$, and also analyze the variance of the estimate. The variance is reduced by taking the mean of multiple independent runs of the estimator.

Let Y_V denote the return value of Algorithm 4. Let p_V denote the number of pairs of butterflies in G that share a single vertex.

Lemma 4. $\mathbb{E}[Y_V] = \bowtie$, *and* $\mathbb{V}\mathrm{ar}[Y_V] \leq \frac{n}{4}(\bowtie + p_V)$

Proof. Consider that the butterflies in G are numbered from 1 to $\bowtie$. Let $X = \bowtie_v$, the number of butterflies that contain the vertex v, which is sampled uniformly. For $i = 1, \ldots, \bowtie$, let X_i be an indicator random variable equal to 1 if the i^{th} butterfly includes the vertex v. We have $X = \sum_{i=1}^{\bowtie} X_i$. Since each butterfly has four vertices, $\mathbb{E}[X_i] = \Pr[X_i = 1] = 4/n$. Thus, $\mathbb{E}[X] = \sum_{i=1}^{\bowtie} \mathbb{E}[X_i] = \sum_{i=1}^{\bowtie} \Pr[X_i = 1] = \frac{4\bowtie}{n}$. Since $Y_V = X \cdot \frac{n}{4}$, we have $\mathbb{E}[Y_V] = \bowtie$.

For the variance of Y_V, we consider the joint probabilities of different butterflies being sampled together. The set of all pairs of butterflies are partitioned into different types as follows. This partitioning will help not only with 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 two 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 $p_V = p_{1v} + p_{2v} + p_{1e} + p_{1w}$ be the number of pairs of butterflies that share at least one vertex.

$$\begin{aligned}\mathbb{V}\mathrm{ar}[Y_V] &= \mathbb{V}\mathrm{ar}\left[\frac{n}{4}\sum_{i=1}^{\bowtie} X_i\right] = \frac{n^2}{16}\mathbb{V}\mathrm{ar}\left[\sum_{i=1}^{\bowtie} X_i\right] \\ &= \frac{n^2}{16}\left[\sum_{i=1}^{\bowtie}\mathbb{V}\mathrm{ar}[X_i] + \sum_{i\neq j}\mathbb{C}\mathrm{ov}(X_i, X_j)\right] \\ &= \frac{n^2}{16}\left[\bowtie\left(\frac{4}{n} - \frac{16}{n^2}\right) + \sum_{i\neq j}\left(\mathbb{E}[X_iX_j] - \mathbb{E}[X_i]\mathbb{E}[X_j]\right)\right]\end{aligned}$$

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

- Type $0v$, there is zero probability of i and j being counted within $\bowtie_v$, hence $\mathbb{E}[X_iX_j] = 0$. $\mathbb{C}\mathrm{ov}(X_i, X_j) = -16/n^2$.
- Type $1v$, $\mathbb{E}[X_iX_j] = \Pr[X_i = 1]\Pr[X_j = 1|X_i = 1] = (4/n)(1/4) = 1/n$. $\mathbb{C}\mathrm{ov}(X_i, X_j) = 1/n - 16/n^2$.
- Type $2v$, $\mathbb{E}[X_iX_j] = (4/n)(1/2) = 2/n$. $\mathbb{C}\mathrm{ov}(X_i, X_j) = 2/n - 16/n^2$
- Type $1e$, $\mathbb{E}[X_iX_j] = (4/n)(1/2) = 2/n$. $\mathbb{C}\mathrm{ov}(X_i, X_j) = 2/n - 16/n^2$
- Type $1w$, $\mathbb{E}[X_iX_j] = (4/n)(3/4) = 3/n$. $\mathbb{C}\mathrm{ov}(X_i, X_j) = 3/n - 16/n^2$.

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

$$\mathbb{V}\mathrm{ar}[Y_V] \leq \frac{n}{4}\bowtie + \frac{n}{16}p_{1v} + \frac{n}{8}p_{2v} + \frac{n}{8}p_{1e} + \frac{3n}{16}p_{1w} \leq \frac{n}{4}(\bowtie + p_V)$$

□

Algorithm 5: ESamp (single iteration)

Input: A bipartite graph $G = (V, E)$
Output: An estimate of $\bowtie(G)$

Choose an edge e from E uniformly at random.
$\bowtie_e \leftarrow$ eBFC (e, G) `// Algorithm 3`
return $\bowtie_e \cdot m/4$

Let Z be the average of $\alpha = \frac{8n}{\epsilon^2 \bowtie} \left(1 + \frac{p_V}{\bowtie}\right)$ independent instances of Y_V. Using $\mathbb{V}\text{ar}[Z] = \mathbb{V}\text{ar}[Y_V]/\alpha$ and Chebyshev's inequality:

$$\Pr[|Z - \bowtie| \geq \epsilon \bowtie] \leq \frac{\mathbb{V}\text{ar}[Z]}{\epsilon^2 \bowtie^2} = \frac{\mathbb{V}\text{ar}[Y_V]}{\alpha \epsilon^2 \bowtie^2} \leq \frac{n(\bowtie + p_V)}{4\alpha\epsilon^2 \bowtie^2} = \frac{1}{32}$$

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

Lemma 5. *There is an algorithm that uses $\tilde{O}\left(\frac{n}{\bowtie}\left(1 + \frac{p_V}{\bowtie}\right)\right)$ iterations [4] of* VSamp *(Algorithm 4) and yields an (ϵ, δ)-estimator of $\bowtie(G)$ using expected time $\tilde{O}\left(\frac{m\Delta}{\bowtie}\left(1 + \frac{p_V}{\bowtie}\right)\right)$. Expected additional space is $O\left(\frac{m\Delta}{n}\right)$.*

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

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, $p_E = p_{1e} + p_{1w}$.

Lemma 6. $\mathbb{E}[Y_E] = \bowtie$ *and* $\mathbb{V}\text{ar}[Y_E] \leq \frac{m}{4}(\bowtie + p_E)$. *Using $\tilde{O}\left(\frac{m}{\bowtie}\left(1 + \frac{p_E}{\bowtie}\right)\right)$ iterations of* ESamp *yields an (ϵ, δ)-estimator of $\bowtie(G)$ using time $\tilde{O}\left(\frac{m^2\Delta}{n\bowtie}\left(1 + \frac{p_E}{\bowtie}\right)\right)$. The additional space complexity is $O\left(\frac{m\Delta}{n}\right)$.*

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 here, with proofs deferred to the full version [25]. Let Y_W denote the return value of Algorithm 6.

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

Algorithm 6: WSamp (single iteration)

Input: A bipartite graph $G = (V, E)$
Output: An estimate of $\bowtie(G)$

$\wedge \leftarrow \sum_{u \in V} \binom{d_u}{2}$ `// number of wedges in G`
Choose a vertex $u \in V$ with probability $\binom{d_u}{2}/\wedge$
Choose two distinct vertices $v, w \in \Gamma_u$ uniformly at random
$\beta \leftarrow |\Gamma_v \cap \Gamma_w| - 1$ `// # butterflies that has (u, v, w)`
return $\beta \cdot \wedge/4$

	VSamp		ESamp		WSamp	
	$\sqrt{\frac{(\bowtie + p_V)n}{4}}$	error %	$\sqrt{\frac{(\bowtie + p_E)m}{4}}$	error %	$\sqrt{\frac{(\bowtie + p_{1w})h}{4}}$	error %
Deli	2.8×10^{13}	494.9	2.1×10^{12}	38.3	7.7×10^{11}	13.6
Journal	2.3×10^{15}	708.56	2.1×10^{13}	6.4	1.4×10^{14}	99.3
Orkut	5×10^{15}	223.6	2×10^{14}	9.2	2.9×10^{14}	13.3
Web	6.8×10^{16}	3431.5	8.2×10^{13}	4.1	4.3×10^{16}	2194.8
Wiki-en	1.3×10^{16}	6823.6	3.8×10^{13}	19	1.4×10^{15}	703.4

Table 3: Standard deviations of estimators on large graphs. For each method, the theoretical upper bound on the standard deviation is shown in the left column, and the "error", the ratio between the (theoretical) standard deviation and the number of butterflies, is shown in the second column.

Lemma 7. $\mathbb{E}[Y_W] = \bowtie$, *and* $\mathbb{V}\text{ar}[Y_W] \leq \frac{\wedge}{4}(\bowtie + p_{1w})$. *Using $\tilde{O}\left(\frac{\wedge}{\bowtie}\left(1 + \frac{p_{1w}}{\bowtie}\right)\right)$ iterations of WSamp yields an (ϵ, δ)-estimator of $\bowtie(G)$ in time $\tilde{O}\left(\frac{(\Delta + \log n)\wedge}{\bowtie}\left(1 + \frac{p_{1w}}{\bowtie}\right)\right)$ and space $O(n)$.*

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_V where p_V 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 $\bowtie \ll p_V, p_E$ and hence $\bowtie + p_V \approx p_V$ and $\bowtie + p_E \approx p_E$. If two butterflies share an edge, they certainly share a vertex, hence $p_E \leq p_V$. Since it is possible that two butterflies share a vertex but do not share an edge, p_E could be much smaller than p_V. It turns out that in most of these graphs, p_E was much smaller than p_V. On the other hand, the number of vertices in a graph (n) is comparable to the number of edges (m). Typically $m < 10n$, and only in one case (Orkut), we have $m \approx 30n$. Thus, $np_V \ll mp_E$ 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 $e = (v, u) \in E$ in $G = (V, E)$
Output: An estimate of $\bowtie_e$

1 Choose a vertex w from Γ_u uniformly at random
2 Choose a vertex x from Γ_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 $\wedge \cdot p_{1w}$, where $\wedge$ is the number of wedges in the graph ($O(\sum_v d_v^2)$) and p_{1w} is the number of pairs of butterflies that share a wedge. The variance of ESAMP is proportional to mp_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 $\wedge$ 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.

LEMMA 8. $\mathbb{E}[Y_{FE}] = \bowtie_e$ *and* $\mathbb{V}\text{ar}[Y_{FE}] \leq \bowtie_e (d_u \cdot d_v)$ (Proof is deferred to [25]).

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

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 `Deli` 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 $\bowtie(G)$ 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 $\bowtie(G)$. 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 $\bowtie(G)$, after applying a scaling factor. This

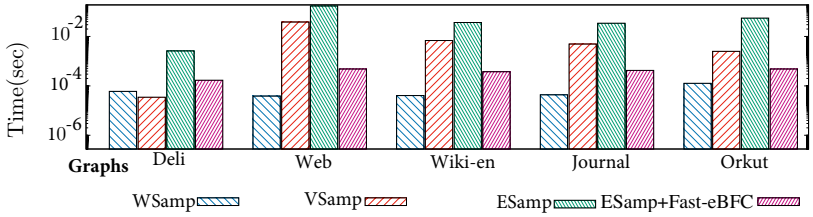


Figure 4: Average time per iteration of sampling algorithms.

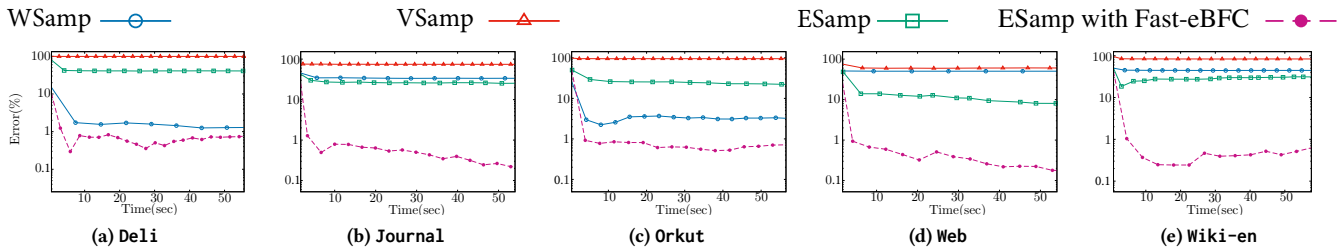


Figure 5: Relative error as a function of runtime, for sampling algorithms. ESamp with Fast-eBFC yields < 1% relative error within 5 seconds for all networks.

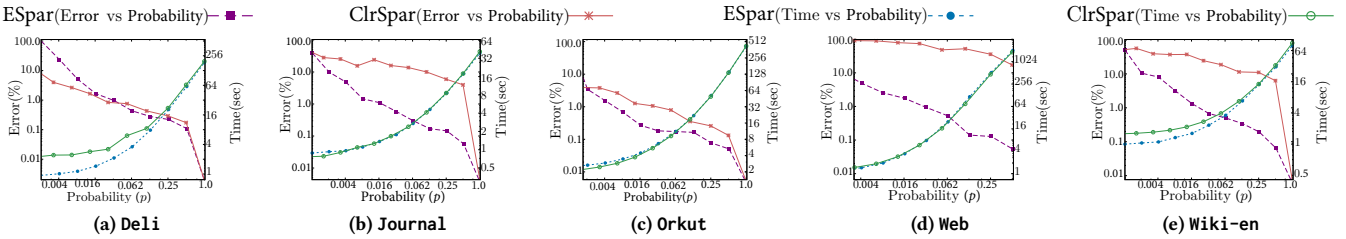


Figure 6: Accuracy (on left y-axis) and runtime performance (on right y-axis) of sparsification algorithms for different probabilities (on x-axis). ESpar performs better than ClrSpar, and yields < 1% relative error within 4 seconds for all networks.

Algorithm 8: ESpar: Edge Sparsification

Input: A bipartite graph $G = (V, E)$, parameter $p, 0 < p < 1$

1 Construct E' by including each edge $e \in E$ independently with probability p

2 $\beta \leftarrow$ ExactBFC(V, E') `// Algorithm 1`

3 **return** $\beta \cdot p^{-4}$

is much faster than working with the original graph G since the number of edges in G' can be much smaller. The ESpar algorithm is described in Algorithm 8.

Lemma 9. *Let Y_{ES} denote the output of* ESpar *on input graph G. Then $\mathbb{E}[Y_{ES}] = \bowtie(G)$. If $p > \max\left(\sqrt[4]{\frac{24}{\bowtie}}, \sqrt{\frac{24\Delta}{\bowtie}}, \frac{24\Delta^2}{\bowtie}\right)$, then $\mathbb{V}\mathrm{ar}[Y_{ES}] \leq \bowtie^2/8$.*

We provide a sketch of the proof below, and the full proof can be found in the full version of the paper [25]. The proof of the expectation 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 that the following types of pairs of butterflies impact the variance: Type $0e$ (share zero edges), Type $1e$ (one edge), and Type $1w$ (one wedge). Let the numbers of such pairs be p_{0e}, p_{1e}, and p_{1w} respectively. 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 Δ and $\bowtie$.

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

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

6.2 Colorful Sparsification (ClrSpar)

The idea in ClrSpar is to sample edges at a rate of p, as in ESpar, 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

Algorithm 9: ClrSpar

Input : Bipartite graph $G = (V, E)$, number of colors N

Let $f : V \to \{1, \ldots, N\}$ // map to random colors

$E' \leftarrow \{(u, v) \in E_G | f(u) = f(v)\}$

$\beta \leftarrow$ ExactBFC (V, E') // Algorithm 1

return $\beta \cdot p^{-3}$ where $p = 1/N$

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 $p^3 \bowtie$, which is higher than in the case of ESpar ($p^4 \bowtie$). 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 $p = 1/N$. $\mathbb{E}[Y] = \bowtie(G)$. If $p > \max(\sqrt[3]{\frac{32}{\bowtie}}, \sqrt{\frac{32\Delta}{\bowtie}}, \frac{32\Delta^2}{\bowtie})$, then $\mathbb{V}\mathrm{ar}[Y] \leq \bowtie^2/8$.*

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 $(\bowtie p^{-3} + p_{2v}p^{-1} - \bowtie p^{-4})$. 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 $\bowtie$. 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.

Graph \ Algorithm	ESpar $\bowtie p^{-4} + p_{1w}p^{-2} + p_{1e}p^{-1}$	ClrSpar $\bowtie p^{-3} + p_{1w}p^{-2} + p_{1e}p^{-1} + p_{2v}p^{-1}$
Deli	4.047×10^{15}	9.163×10^{20}
Journal	1.413×10^{19}	2.042×10^{25}
Orkut	5.576×10^{19}	8.514×10^{25}
Web	7.760×10^{20}	4.692×10^{27}
Wiki-en	1.303×10^{20}	3.062×10^{26}

Table 4: Upper bounds on ESpar & ClrSpar variances, p=0.1.

	ESamp (with Fast-eBFC)	ESpar
Deli	3.4	2.1
Journal	5.0	1.7
Orkut	3.4	3.4
Web	4.1	3.9
Wiki-en	4.8	2.3

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 ≈ 4 and ≈ 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

[1] 2017. GNIP. (https://gnip.com/about/).
[2] 2017. The Graph API for the Facebook Social Graph. (https://developers.facebook.com/docs/graph-api).
[3] N. K. Ahmed, J. Neville, R. A. Rossi, N. Duffield, and T. L. Willke. 2016. Graphlet Decomposition: Framework, Algorithms, and Applications. *KAIS* (2016), 1–32.
[4] S. Aksoy, T. G. Kolda, and A. Pinar. 2017. Measuring and Modeling Bipartite Graphs with Community Structure. *Journal of Complex Networks* 5, 4 (2017), 581–603.
[5] N. Alon, R. Yuster, and U. Zwick. 1997. Finding and counting given length cycles. *Algorithmica* 17, 3 (1997), 209–223.
[6] Stephen P. Borgatti and Martin G. Everett. 1997. Network analysis of 2-mode data. *Social Networks* 19, 3 (1997), 243 – 269.
[7] Marco Bressan, Flavio Chierichetti, Ravi Kumar, Stefano Leucci, and Alessandro Panconesi. 2017. Counting Graphlets: Space vs Time. In *WSDM*. 557–566.
[8] L. De Stefani, A. Epasto, M. Riondato, and E. Upfal. 2016. TRIÈST: Counting Local and Global Triangles in Fully-Dynamic Streams with Fixed Memory Size. In *KDD*. 825–834.
[9] T. R. Halford and K. M. Chugg. 2006. An algorithm for counting short cycles in bipartite graphs. *IEEE Transactions on Information Theory* 52, 1 (2006), 287–292.
[10] Alon Itai and Michael Rodeh. 1978. Finding a Minimum Circuit in a Graph. *SIAM J. Comput.* 7, 4 (1978), 413–423.
[11] Shweta Jain and C. Seshadhri. 2017. A Fast and Provable Method for Estimating Clique Counts Using Turán's Theorem. In *WWW*. 441–449.
[12] Madhav Jha, C. Seshadhri, and Ali Pinar. 2015. Path Sampling: A Fast and Provable Method for Estimating 4-Vertex Subgraph Counts. In *WWW*. 495–505.
[13] M. Latapy, C. Magnien, and N. Del Vecchio. 2008. Basic notions for the analysis of large two-mode networks. *Social Networks* 30, 1 (2008), 31 – 48.
[14] Yongsub Lim and U Kang. 2015. MASCOT: Memory-efficient and Accurate Sampling for Counting Local Triangles in Graph Streams. In *KDD*. 685–694.
[15] Pedro G. Lind, Marta C. González, and Hans J. Herrmann. 2005. Cycles and clustering in bipartite networks. *Phys. Rev. E* 72 (2005), 056127. Issue 5.
[16] R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. 2002. Network Motifs: Simple Building Blocks of Complex Networks. *Science* 298, 5594 (2002), 824–827.
[17] M.E.J. Newman. 2001. Scientific collaboration networks. II. Shortest paths, weighted networks, and centrality. *Phys. Rev. E* 64 (2001), 016132. Issue 1.
[18] M. E. J. Newman. 2001. Scientific collaboration networks. I. Network construction and fundamental results. *Phys. Rev. E* 64 (2001), 016131. Issue 1.
[19] Tore Opsahl. 2013. Triadic closure in two-mode networks: Redefining the global and local clustering coefficients. *Social Networks* 35, 2 (2013), 159 – 167. Special Issue on Advances in Two-mode Social Networks.
[20] Rasmus Pagh and Charalampos E Tsourakakis. 2012. Colorful triangle counting and a mapreduce implementation. *Inform. Process. Lett.* 112, 7 (2012), 277–281.
[21] A. Pavan, K. Tangwongsan, S. Tirthapura, and K. Wu. 2013. Counting and Sampling Triangles from a Graph Stream. *PVLDB* 6, 14 (2013), 1870–1881.
[22] Ali Pinar, C. Seshadhri, and Vaidyanathan Vishal. 2017. ESCAPE: Efficiently Counting All 5-Vertex Subgraphs. In *WWW*. 1431–1440.
[23] 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.
[24] S.-V. Sanei-Mehri, A. Erdem Sariyuce, and S. Tirthapura. 2018. Butterfly Counting in Bipartite Networks. https://github.com/beginner1010/butterfly-counting.
[25] S.-V. Sanei-Mehri, A. Erdem Sariyuce, and S. Tirthapura. 2018. Butterfly Counting in Bipartite Networks. *ArXiv e-prints* (Dec. 2018). arXiv:1801.00338
[26] Ahmet Erdem Sarıyüce and Ali Pinar. 2018. Peeling Bipartite Networks for Dense Subgraph Discovery. In *WSDM*.
[27] C. Seshadhri, A. Pinar, and T. G. Kolda. 2014. Triadic Measures on Graphs: The Power of Wedge Sampling. *Statistical Analysis and Data Mining* 7, 4 (2014), 294–307.
[28] K. Tangwongsan, A. Pavan, and S. Tirthapura. 2013. Parallel Triangle counting in massive streaming graphs. In *CIKM*. 781–786.
[29] Charalampos E Tsourakakis, U Kang, Gary L Miller, and Christos Faloutsos. 2009. Doulion: counting triangles in massive graphs with a coin. In *KDD*. 837–846.
[30] D. Turkoglu and A. Turk. 2017. Edge-Based Wedge Sampling to Estimate Triangle Counts in Very Large Graphs. In *2017 IEEE ICDM*. 455–464.
[31] J. Wang, A. W. C. Fu, and J. Cheng. 2014. Rectangle Counting in Large Bipartite Graphs. In *2014 IEEE International Congress on Big Data*. 17–24.
[32] B. Wu, K. Yi, and Z. Li. 2016. Counting Triangles in Large Graphs by Random Sampling. *IEEE TKDE* 28, 8 (2016), 2013–2026.