Scalable Approximate Butterfly and Bi-triangle Counting for Large Bipartite Networks

FANGYUAN ZHANG, The Chinese University of Hong Kong, Hong Kong SAR
DECHUANG CHEN*, The Chinese University of Hong Kong, Hong Kong SAR
SIBO WANG†, The Chinese University of Hong Kong, Hong Kong SAR
YIN YANG, Hamad Bin Khalifa University, Qatar
JUNHAO GAN, The University of Melbourne, Australia

A *bipartite* graph is a graph that consists of *two disjoint sets* of vertices and only edges between vertices from different vertex sets. In this paper, we study the counting problems of two common types of *motifs* in bipartite graphs: (i) *butterflies* (2×2 bicliques) and (ii) *bi-triangles* (length-6 cycles). Unlike most of the existing algorithms that aim to obtain exact counts, our goal is to obtain *precise enough estimations* of these counts in bipartite graphs, as such estimations are already sufficient and of great usefulness in various applications. While there exist approximate algorithms for butterfly counting, these algorithms are mainly based on the techniques designed for general graphs, and hence, they are less effective on bipartite graphs. Not to mention that there is still a lack of study on approximate bi-triangle counting.

Motivated by this, we first propose a novel butterfly counting algorithm, called *one-sided weighted sampling*, which is tailored for bipartite graphs. The basic idea of this algorithm is to estimate the total butterfly count with the number of butterflies containing two *randomly sampled* vertices from the same side of the two vertex sets. We prove that our estimation is *unbiased*, and our technique can be further extended (non-trivially) for bi-triangle count estimation. Theoretical analyses under a *power-law random bipartite graph model* and extensive experiments on multiple large real datasets demonstrate that our proposed approximate counting algorithms can reach high accuracy, yet achieve up to three orders (resp. four orders) of magnitude speed-up over the state-of-the-art exact butterfly (resp. bi-triangle) counting algorithms. Additionally, we present an approximate clustering coefficient estimation framework for bipartite graphs, which shows a similar speed-up over the exact solutions with less than 1% relative error.

CCS Concepts: • **Theory of computation → Graph algorithms analysis**.

Additional Key Words and Phrases: Graph Algorithms; Approximation Algorithms; Sampling

ACM Reference Format:
Fangyuan Zhang, Dechuang Chen, Sibo Wang, Yin Yang, and Junhao Gan. 2023. Scalable Approximate Butterfly and Bi-triangle Counting for Large Bipartite Networks. *Proc. ACM Manag. Data* 1, 4 (SIGMOD), Article 259 (December 2023), 26 pages. https://doi.org/10.1145/3626753

*Part of the work was done while Dechuang Chen is at Xidian University.
†Sibo Wang is the corresponding author.

Authors' addresses: Fangyuan Zhang, fzhang@se.cuhk.edu.hk, The Chinese University of Hong Kong, Hong Kong SAR; Dechuang Chen, dcchen@se.cuhk.edu.hk, The Chinese University of Hong Kong, Hong Kong SAR; Sibo Wang, swang@se.cuhk.edu.hk, The Chinese University of Hong Kong, Hong Kong SAR; Yin Yang, yyang@hbku.edu.qa, Hamad Bin Khalifa University, Qatar; Junhao Gan, junhao.gan@unimelb.edu.au, The University of Melbourne, Australia.

2836-6573/2023/12-ART259 $15.00
https://doi.org/10.1145/3626753

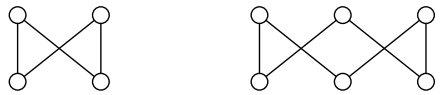
1 INTRODUCTION

Bipartite graph, or network, is a commonly used data structure in practice, with numerous real-world applications [10, 38]. Typically, such a graph represents the relationships between entities from two disjoint groups. For instance, in biology [65], such two groups can be genes and functions (i.e., each edge in the bipartite graph indicates a connection between a gene and a function), or diseases and effective drugs (where an edge signifies that a drug is effective for treating a particular disease). Similarly, in e-commerce, a bipartite graph can be constructed with two different sets of vertices, for users and products respectively, with edges representing purchases. Other use cases include user ratings [89], text [39] and authorship [63] networks, etc. Bipartite graphs are commonly found in research: as evidence, more than half of the networks in the well-established Konect network collection [36] are bipartite networks, which have been used in a plethora of research works [11, 17, 18, 21, 26, 29, 45, 80] focusing on bipartite graphs in diverse fields.

As shown in the literature, counting the number of different kinds of motifs in a graph is an important approach to deriving graph-level features, which can be used for graph level tasks like graph classification [59] and community search/analysis [25, 30, 69]. Moreover, as shown in the literature, deriving an approximate estimation of such a number of motifs is sufficient and existing solutions generally adopt sampling to derive the estimations of the number of motifs [59]. Among counting different motifs, one of the most important tasks is to count *triangles* (i.e., 3-cliques), which is the smallest type of clique in general graphs and plays an important role in various network analyses such as clustering coefficient computation [58], social network analysis [20, 34], community detection [19, 30, 31], etc. Yet, these techniques are devised for general graphs and discard the special properties of bipartite graphs.

This paper focuses on estimating the number of basic motifs in a large bipartite graph. For a bipartite graph, the concept of *butterfly* is of significant importance, comparable to that of the triangle in unipartite graphs. A butterfly is a 2×2 bi-clique, as illustrated in Fig. 1(a). It is among the simplest motif types in a bipartite graph and has been used in definitions of graph metrics such as *bipartite clustering coefficient*, a cohesiveness measure for bipartite networks. In particular, given a bipartite graph G, its bipartite clustering coefficient [5, 41, 48, 53] can be defined as $4 \times \bowtie_G / \wedge_G$, where $\bowtie_G$ denotes the number of butterflies in G, and $\wedge_G$ is the number of *caterpillars* (i.e., 3-paths) in G. A major bottleneck for bipartite clustering coefficient computation is counting the number of butterflies. Apart from its use in computing bipartite clustering coefficients, the butterfly count itself can be a meaningful metric that captures relationship statistics between vertices. For instance, in a director-company bipartite network [49, 53], a butterfly means two directors meet on multiple boards; hence, the larger the number of butterflies, the more intertwined the companies are.

Another popular bipartite motif is the *bi-triangle* [48, 84], shown in Fig. 1(b), which is a 6-cycle containing three vertices in one vertex set L of the bipartite graph G, and another three vertices in the other set R. Similar to the case of butterflies, bi-triangles have been used in an alternative definition of bipartite clustering coefficient [48, 84]. One advantage of this alternative definition is that it can be adapted to measure the cohesiveness of a specific side of the graph. In particular, given a vertex set (say, L) in G, its corresponding *one-sided clustering coefficient* based on bi-triangle can be defined as $3 \times \bowtie_G / \wedge_{G,L}$, where $\bowtie_G$ denotes the number of bi-triangles in G, and $\wedge_{G,L}$ is the number of 4-paths centered at a vertex in L. The other side of the clustering coefficient can be defined similarly based on R. Note that it is unclear how (and whether) this can be done with the butterfly-based definition since the latter involves 3-paths that do not have a center vertex. In addition, as shown in previous work [84], the bi-triangle-based clustering coefficient can be used to measure the quality of clustering/community results returned by a variety of community detection



(a) Example butterfly (b) Example bi-triangle

Fig. 1. Bipartite graph motifs: butterfly and bi-triangle

algorithms (e.g., (α, β)-core [42], k-wing [77], and k-Ctruss [83]) and clustering methods (e.g., BLP [6] and BRIM [43]).

Given the important applications of butterfly and bi-triangle counting, it is not surprising that much research effort [72, 74, 84] has been devoted to performing such counting efficiently. However, for sizable graphs, the cost of computing the exact counts of these motifs remains prohibitive. Further, in real applications, butterfly or bi-triangle counting may be executed more than once on different sub-graphs of the input bipartite graph [74, 84]. For example, the community results returned by different algorithms might need to compute and compare the clustering coefficient to measure the cohesiveness of the corresponding result [73, 84]. Luckily, according to existing studies [59], obtaining an accurate estimate of such counts is sufficient in practice, which promises far lower overhead. For example, an accurate estimate of such counts can be used to calculate an accurate estimation of clustering coefficients. More discussion about the application scope of approximate counting can be found in Sec. 8. A notable work in this direction is by Sanei et al. [55], which investigates fast approximate butterfly counting via local sampling, e.g., vertex sampling, edge sampling, and wedge sampling. As we explain in Sec. 2.2, all these algorithms share the same design philosophy: adapting approximate triangle counting solutions on general graphs to the new problem of butterfly counting. As we point out, none of these methods effectively utilizes the properties of bipartite graphs, leading to missed opportunities for further cost reductions. In addition, the methods [55] are limited to butterfly counting, and it is unclear how to adapt them to bi-triangles. Consequently, practitioners are left with exact counting solutions for bi-triangles, which incur hefty costs for large graphs.

Motivated by this, we propose effective and efficient algorithms for approximate butterflies and bi-triangles counting, which exploit the special properties of bipartite graphs. In particular, for butterfly counting, we present a *one-sided* local sampling algorithm, utilizing the property that a bipartite graph is divided into two disjoint sets of vertices. Then, given two randomly sampled vertices u and v from one side of the vertex set, say L, we count the number of common neighbors of u and v and derive the number of butterflies that involve u and v accordingly. Counting common neighbors is a simple and intuitive idea and has been applied to other contexts, e.g., [61, 72]. Our main contribution is that we apply this idea to the problems of approximate butterfly/bi-triangle counting, with non-trivial algorithmic designs and analyses. First, we devise *pair sampling* and show how to derive an unbiased estimator for butterfly counting with fine-grained variance analysis. Second, based on the intuition that vertices with larger degrees generally form more butterflies, which amplifies the efficiency gain of our one-sided sampling method, we devise a refined *weighted one-sided pair sampling* algorithm that samples vertices proportionally to their degrees. The estimation result is carefully normalized to maintain its unbiasedness. Moreover, based on a power-law random graph model, we prove that the proposed solutions achieve lower asymptotically costs compared to the state-of-the-art solution [55].

Going one step further, we present a non-trivial extension of our weighted one-sided sampling algorithm to bi-triangle counting, with rigorous analysis of its effectiveness (in terms of unbiasedness and variance) and efficiency (costs under our power-law random graph model). Extensive

experiments conducted on multiple large-scale real bipartite networks validate the high accuracy and efficiency of the proposed algorithms. With the proposed approximate butterfly and bi-triangle counting algorithms, we further present an approximate clustering coefficient estimation framework for bipartite graphs. Experimental results demonstrate that our framework achieves up to three (resp. four) orders of magnitude speedup over the state-of-the-art exact algorithm that computes the butterfly-based (resp. bi-triangle-based) clustering coefficient while providing high-quality estimation with a relative error of less than 1%. In sum, the main contributions of this paper include:

- A novel weighted, one-sided pair sampling algorithm for unbiased approximate butterfly counting that exploits properties of the bipartite graph.
- Analysis of the proposed solution under a power-law random bipartite graph model, which shows that our algorithm achieves a lower asymptotic time complexity compared to the current state-of-the-art, under the same expected error.
- A non-trivial extension of our sampling scheme to bi-triangle counting, with rigorous analysis of its unbiasedness, variance, and time complexity.
- An approximation framework for butterfly-based and bi-triangle-based clustering coefficient.
- Extensive experiments involving large real bipartite graphs, whose results confirm that the proposed methods achieve significant performance gains in terms of running time compared to existing solutions, under comparable levels of result accuracy.

2 BACKGROUND

2.1 Preliminaries

We consider an unweighted and undirected bipartite graph $G = \langle V = (L \cup R), E\rangle$, where each pair of vertices can have at most one edge connecting them. A bipartite graph has the following special properties: (i) The vertex set V is partitioned into two disjoint subsets L and R, i.e., $L \cap R = \emptyset$; (ii) The edge set $E \subseteq L \times R$, where $\times$ means the Cartesian product, that is, the edges only exist between vertices from different sets. An edge between two vertices u and v is denoted as (u, v). We use $N[v] = \{u|(v, u) \in E\}$ to denote the set of neighbors of vertex v, and $d_v = |N[v]|$ for the degree of v. In addition, we define $\xi_{u,v}$ to denote the size of the intersection of the neighbors of u and v, i.e., $\xi_{u,v} = |N[u] \cap N[v]|$. Similarly, $\xi_{u,v,w}$ denotes the size of the intersection of the neighbors of u, v, and w. Let m and n be the sizes of the edge set and the vertex set, respectively. We define the butterfly motif as follows.

Definition 2.1 (Butterfly). Given a bipartite graph $G = \langle V = (L \cup R), E\rangle$ and four vertices $a, b, c, d \in V$ where $a, b \in L$ and $c, d \in R$. When edges $(a, c), (b, c), (a, d), (b, d)$ are all in E, the induced subgraph of $\{a, b, c, d\}$ is a butterfly.

We use $\bowtie_G$ ($\bowtie$ when G is clear from the context) to denote the number of butterflies in G, and $\bowtie_v$ to mean the number of butterflies containing vertex v. Given a subgraph S, $\bowtie_S$ represents the number of butterflies that contain all vertices in S. Next, we define another important motif: bi-triangle.

Definition 2.2 (Bi-triangle). Given a bipartite graph $G = \langle V = (L \cup R), E\rangle$, a bi-triangle is a cycle containing 6 vertices $a, b, c, d, e, f \in V$, where three of them are in L and the other three are in R, and edges $(a, b), (b, c), (c, d), (d, e), (e, f), (f, a) \in E$. We use the notation *a-b-c-d-e-f-a* to represent such a bi-triangle.

Similar to the case of butterflies, we use $\bowtie_G$ (or simply $\bowtie$ when G is clear from the context) to denote the bi-triangle count in G, $\bowtie_v$ for the number of bi-triangles containing vertex v, and $\bowtie_S$ for the number of bi-triangles covering all vertices in subgraph S.

Definition 2.3 (Wedge). Given graph $G = \langle V, E\rangle$, a wedge is a 3-path consisting of three arbitrary vertices $a, b, c \in V$ and two edges (a, b) and (b, c) connecting them.

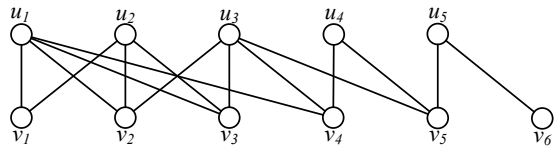


Fig. 2. An example of a bipartite graph

Example 2.4. As an example, Fig. 2 shows a bipartite graph G, in which $L = \{u_1, u_2, u_3, u_4, u_5\}$ and $R = \{v_1, v_2, v_3, v_4, v_5, v_6\}$. Vertices u_1, v_1, u_2 form a wedge, and vertices u_1, u_2, v_1, v_2 form a butterfly. Meanwhile, there are two bi-triangles, both containing the same set of vertices $u_1, u_2, u_3, v_1, v_2, v_3$: u_1-v_1-u_2-v_2-u_3-v_3-u_1 and u_1-v_1-u_2-v_3-u_3-v_2-u_1. For the whole bipartite graph G, we have $\boxtimes = 8$ and $\bowtie = 8$. If we restrict our focus to a particular set of vertices, say, $S = \{u_1, u_2\}$, then $\boxtimes_{u_1,u_2} = 3$.

Frequently used notations are summarized in Tab. 1. Following previous work [55], when estimating the number of motifs, we aim to provide estimates that satisfy the following definition. Such a definition is also widely used in other studies, like personalized PageRank [27, 78, 79], graph clustering [54, 86], influence maximization [22, 23, 66], and data analytics [13, 14, 40, 46].

Definition 2.5 ($(\epsilon, \delta)-$approximation). Given parameters $\epsilon, \delta \in [0, 1]$, an estimation $\hat{Z}$ of a variable Z is an (ϵ, δ)-approximation of Z if $Pr[|\hat{Z} - Z| > \epsilon \cdot |Z|] \leq \delta$.

2.2 State of the Art

Butterfly Counting. Since a butterfly is essentially a cycle of length 4, the total number of butterflies in a bipartite graph G can reach $O(m^2)$, where m denotes the number of edges in G. Consequently, when m is large, simply enumerating all possible butterflies in G incurs prohibitive costs. Wang et al. [72] present an optimized algorithm for exact butterfly counting that avoids the exhaustive enumeration of all butterflies. However, this method still has a rather high computational overhead, as shown in the previous work [72]. Wang et al. [76] propose a vertex-priority-based exact butterfly counting method, which improves upon the previous layer-priority-based algorithm, and further exploits a cache-aware policy to reduce costs. However, their algorithm still demands $O(m^{1.5})$ running time and takes close to 1,000 seconds (i.e., 16 minutes) to complete on a large graph of around half a billion edges according to their experiments. As a result, for massive graphs and applications that do not necessitate exact counting values, approximate butterfly counting algorithms are more suitable due to their higher efficiency.

Sanei et al. [55] propose to estimate the number of butterflies in a given $G = \langle V = (L, R), E \rangle$ through *local sampling*, which includes vertex sampling, edge sampling, and wedge (see Definition 2.3) sampling. Take vertex sampling for example. First, the method samples a vertex $u \in V$ uniformly at random. Then, it computes the exact number of butterflies containing u, and subsequently multiplies this count by $n/4$ to obtain an unbiased estimate of the total number of butterflies in G. Note that this method does not utilize the fact that G is a bipartite graph. In particular, vertex u is sampled from the combined vertex set V, regardless of whether u comes from L or R. Similarly, in their edge sampling and wedge sampling approaches, the fact that the given graph G is bipartite is largely irrelevant. Their experiments show that a variant of edge sampling, referred to as *FastEdge*, is the most performant among all their local sampling techniques.

Bi-triangle Counting. Yang et al. [84] investigate the problem of exact bi-triangle counting on large bipartite networks. Among their solutions, the most efficient one is RSWJ-count, which represents each bi-triangle as a so-called RSWJ-unit. In addition, they design two efficient algorithms, V-LCount and E-LCount, for counting bi-triangles containing a given vertex or edge, respectively. Unfortunately, RSWJ-count still takes over 10,000 seconds (i.e., 2.7 hours) on a large bipartite graph,

Table 1. Frequently used notations.

Notation	Description
$G = \langle V, E \rangle$	a bipartite graph that consists of vertex set V and edge set E
n, m	the number of nodes and edges, respectively
L, R	two disjoint vertex subsets of V
$e = (u, v)$	an edge e between vertices u and v
$N[v], d_v$	the set of neighbors of v, the degree of v
ξ_S	the intersection size of neighbors for each vertex in S
$\bowtie_S$	the number of butterflies contain all vertices in S
$\bowtie_G$	the number of butterflies in G
$\bowtie\!\!\!\!\wedge_S$	the number of bi-triangles contain all vertices in S
$\bowtie\!\!\!\!\wedge_G$	the number of bi-triangles in G

according to the experiments of previous work [84]. To our knowledge, there does not yet exist an efficient and accurate solution for approximate bi-triangle counting.

3 APPROXIMATE BUTTERFLY COUNTING

This section presents our novel local sampling algorithms for approximate butterfly counting in a bipartite graph G. In general, the workflow of local sampling is as follows: (i) start with a set of subgraphs of the same type, (ii) sample an element S from this set, (iii) compute the number of butterflies $\bowtie_S$ containing S, and (iv) derive an unbiased estimator of the total number of butterflies $\bowtie_G$, based on $\bowtie_S$ and the size of the subgraph set in Step (i). Observe that Step (iii) is key to the overall efficiency of the above framework since it is this step that accesses the graph structure of G and computes a butterfly count, which is usually the most time-consuming part of the above workflow. Ideally, given a subgraph S drawn in Step (ii), Step (iii) should quickly obtain the local butterfly count $\bowtie_S$, *without enumerating the* $\bowtie_S$ *individual butterflies*.

Regarding the accuracy of the local sampling algorithm, assuming that Step (iii) returns an exact count $\bowtie_S$, and Step (iv) successfully obtains an unbiased estimate for $\bowtie$, then the main source of error is *sample variance*, which depends on the type of the subgraph S. For instance, intuitively setting S to be a single vertex (as is done in the vertex sampling approach [55]) is not a good idea from the perspective of minimizing sample variance, since a "popular" vertex may form numerous butterflies, whereas an isolated vertex may not form any butterfly at all, leading to high variance. A larger sampling unit S, such as an edge or a wedge, generally leads to a more stable local count, as shown in the previous work [55].

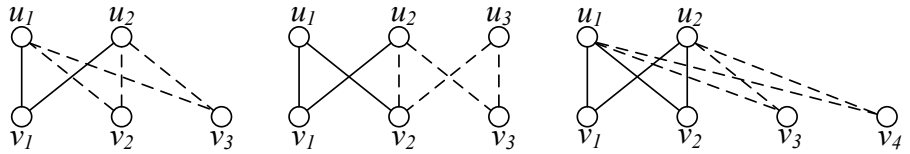
Following the above intuitions, we design an effective and efficient solution *one-sided pair sampling*, presented in Sec. 3.1, which performs fast local counting by capturing many butterflies at once, and uses a larger sampling unit (i.e., a vertex pair) with relatively stable local counts. Then, we further apply the heuristic that vertices with larger degrees generally form more butterflies, which leads to an optimized *weighted one-sided pair sampling* algorithm, described in Sec. 3.2.

3.1 One-Sided Pair Sampling

Our proposed algorithm, one-sided pair sampling, exploits the fact that vertices in the input graph $G = \langle V = (L, R), E \rangle$ are split into two "sides", i.e., L and R. The sampling unit S is a pair of vertices u, v *on the same side*. Without loss of generality, in the following, we assume that $u, v \in L$, and defer the discussion on how to choose the appropriate sampling side to Sec. 7. An immediate benefit of setting the sampling unit S to a same-side vertex pair is that the corresponding local count $\bowtie_S$

Algorithm 1: One-Sided-Pair-Sampling(G)

1 Sample a pair u, v ($u \neq v$) from L uniformly at random;
2 $\boxtimes_{u,v} \leftarrow \binom{\xi_{u,v}}{2}$;
3 **return** $\binom{|L|}{2} \cdot \boxtimes_{u,v}$;



(a) Type 1 (b) Type 2 (c) Type 3

Fig. 3. Three types of butterfly pairs

(which is $\boxtimes_{u,v}$ in our case) can be efficiently computed by the intersection of the neighbors of u and v, i.e., $\boxtimes_{u,v} = \binom{\xi_{u,v}}{2}$ without enumerating individual butterflies [72].

According to the above method, the number of butterflies containing the pair u, v is a quadratic function of $\xi_{u,v}$. The set intersection $N[u] \cap N[v]$ can be performed, e.g., using a hash-based algorithm. Therefore, *with linear costs of computing $N[u] \cap N[v]$, pair sampling captures a quadratic number of butterflies*. Thus, our local counting step is highly efficient. Alg. 1 shows the pseudo-code for pair sampling, which returns an unbiased estimation of $\boxtimes$. Here, we briefly explain the intuition behind the scaling operation in Line 3 of Alg. 1. Assume that the $\boxtimes$ butterflies in the input graph G are numbered from 1 to $\boxtimes$. We use X_i to indicate whether the i-th butterfly contains u and v, i.e., X_i is 1 when the butterfly contains u and v, and 0 otherwise. Let $X = \boxtimes_{u,v} = \sum_{i=1}^{\boxtimes} X_i$ be the random variable for the number of butterflies contain u and v. Because there are $\binom{|L|}{2}$ possible vertex pairs of $u, v \in L$, we have $\Pr[X_i = 1] = 1/\binom{|L|}{2}$. In more detail, any butterfly in a bipartite graph must contain two vertices from side L. Consider the i-th butterfly, and assume that the left side vertices of this butterfly are u and v. Then, the probability that the randomly sampled two vertices from L are exactly u and v is $1/\binom{|L|}{2}$. According to our sampling strategy, as long as both u and v are chosen, $X_i = 1$ holds; otherwise, $X_i = 0$. By the linearity of expectation, we have $\mathbb{E}[X] = \sum_{i=1}^{\boxtimes} \mathbb{E}[X_i] = \boxtimes/\binom{|L|}{2}$. Hence, by multiplying our local count $\boxtimes_{u,v}$ by $\binom{|L|}{2}$, we obtain an unbiased estimator for $\boxtimes$. Let Y_P denote the estimation result of Alg. 1. Then, we arrive at the following lemma:

Lemma 3.1. $\mathbb{E}[Y_P] = \boxtimes$.

Next, we derive the variance of Y_P. Let $P_{2,L}$ be the number of butterfly pairs (i.e., two different butterflies in G) that share two common vertices in L. For example, in Fig. 3, suppose that $L = \{u_1, u_2, u_3\}$ and $R = \{v_1, v_2, v_3\}$. Then, in Fig. 3(a), the two butterflies containing $\{u_1, u_2, v_1, v_2\}$ and $\{u_1, u_2, v_2, v_3\}$ respectively share the same two vertices $u_1, u_2 \in L$. Similarly, the two butterflies in Fig. 3(c), shown in solid and dashed lines, share the same two vertices $u_1, u_2 \in L$. Thus, these two types of butterflies are counted towards $P_{2,L}$. On the other hand, the same cannot be said for the two butterflies in Fig. 3(b), with vertex sets $\{u_1, u_2, v_1, v_2\}$ and $\{u_2, u_3, v_2, v_3\}$ respectively, since they only overlap on one vertex u_2. We have the following result on the variance of pair sampling.

Lemma 3.2. $\mathbb{V}\mathrm{ar}[Y_P] \leq \binom{|L|}{2}(\boxtimes + p_{2,L})$.

Proof. The variance of Y_P can be analyzed by the sum of X_i, which can be written as follows:

$$\mathbb{V}\mathrm{ar}[Y_P] = \mathbb{V}\mathrm{ar}[\binom{|L|}{2}\sum_{i=1}^{\boxtimes} X_i] = \binom{|L|}{2}^2 \mathbb{V}\mathrm{ar}[\sum_{i=1}^{\boxtimes} X_i] = \binom{|L|}{2}^2 \left(\sum_{i=1}^{\boxtimes} \mathbb{V}\mathrm{ar}[X_i] + \sum_{i\neq j} \mathbb{C}\mathrm{ov}\left(X_i, X_j\right)\right)$$

$$= \binom{|L|}{2}^2 \left(\boxtimes\left(\mathbb{E}[X_i^2] - \mathbb{E}[X_i]^2\right) + \sum_{i\neq j}\left(\mathbb{E}[X_iX_j] - \mathbb{E}[X_i]\mathbb{E}[X_j]\right)\right)$$

$$= \binom{|L|}{2}^2 \left(\boxtimes\left(\frac{1}{\binom{|L|}{2}} - \frac{1}{\binom{|L|}{2}^2}\right) + \sum_{i\neq j}\left(\mathbb{E}[X_iX_j] - \mathbb{E}[X_i]\mathbb{E}[X_j]\right)\right).$$

Consider the butterfly pair (i, j): If butterfly i and j do not share any vertex, it is impossible for $\boxtimes_{u,v}$ to compute butterfly i and j at the same time. In this case, $\mathbb{E}[X_iX_j] = 0, \mathbb{C}\mathrm{ov}\left(X_i, X_j\right) = -1/\binom{|L|}{2}^2$. For the case where i and j share only one vertex and one edge, the corresponding $\mathbb{C}\mathrm{ov}\left(X_i, X_j\right)$ is the same as above. Next, we consider that butterfly i and j both contain u, v (refer to Fig. 3(a)and(c)), in which case the vertices on the other side may be three or four. In both cases, their covariances are the same and can be computed as follows: $\mathbb{E}[X_iX_j] = \Pr[X_i = 1]\Pr[X_j = 1|X_i = 1] = 1/\binom{|L|}{2}\cdot 1 = 1/\binom{|L|}{2}$, $\mathbb{C}\mathrm{ov}\left(X_i, X_j\right) = 1/\binom{|L|}{2} - 1/\binom{|L|}{2}^2$. Let $P_{2,L}$ denote the number of butterfly pairs that share the same two vertices in side L. Then, $\mathbb{V}\mathrm{ar}[Y_P]$ can be written as:

$$\binom{|L|}{2}^2 \left(\boxtimes\left(\frac{1}{\binom{|L|}{2}} - \frac{1}{\binom{|L|}{2}^2}\right) + P_{2,L}\cdot\frac{1}{\binom{|L|}{2}} - \binom{\boxtimes}{2}\cdot\frac{1}{\binom{|L|}{2}^2}\right) \leq \binom{|L|}{2}\left(\boxtimes + P_{2,L}\right) - \binom{\boxtimes}{2} \leq \binom{|L|}{2}\left(\boxtimes + P_{2,L}\right).$$

This finishes the proof. □

To reduce the variance of the estimate, we apply the standard approach of running the proposed pair sampling algorithm multiple times (say, t runs), each with an independent sample, and averaging their results. Let $\bar{Y}_P$ be the mean value of t independent runs. By Chebyshev's inequality, we have

$$\Pr[|\bar{Y}_P - \boxtimes| \geq \epsilon\boxtimes] \leq \frac{\mathbb{V}\mathrm{ar}[\bar{Y}_P]}{\epsilon^2\boxtimes^2} \leq \frac{\binom{|L|}{2}(\boxtimes + p_{2,L})}{t\epsilon^2\boxtimes^2} = p_f,$$

where ϵ is a multiplicative error bound, and the failure probability p_f is defined as the right-hand side of the above inequality. In practice, both ϵ and p_f are usually fixed to small constants. This leads to the following result, in which m is the number of edges in G, and ϵ, p_f are considered constants in the big-O notation.

Lemma 3.3. *By setting* $t = \frac{\binom{|L|}{2}(\boxtimes + p_{2,L})}{p_f\cdot\epsilon^2\boxtimes^2} = O\left(\frac{|L|^2}{\boxtimes}\left(1 + \frac{P_{2,L}}{\boxtimes}\right)\right)$*, we obtain an* ϵ*-error estimate for the number of butterflies* $\boxtimes$ *in* $O\left(\frac{|L|m}{\boxtimes}\left(1 + \frac{P_{2,L}}{\boxtimes}\right)\right)$ *expected time with* $1 - p_f$ *probability.*

Proof. We choose L as the sampled side. First, calculate the size of the intersection of neighbors of u and neighbors of v via hashing. The algorithm runs in $O\left(d_u + d_v\right)$. Hence, the expected time for each sample is $= O\left(\bar{d_L}\right)$, where $\bar{d_L}$ is the average degree of L, which equals $\frac{m}{|L|}$. Since we need to sample $t = \frac{\binom{|L|}{2}(\boxtimes + P_{2,L})}{p_f\epsilon^2\boxtimes^2}$ times, the total expected time complexity is $O\left(\frac{|L|m}{\boxtimes}\left(1 + \frac{P_{2,L}}{\boxtimes}\right)\right)$. □

To further reduce the failure probability p_f, we can apply the median trick [33], which runs c sets of sampling, each with t independent samples as described above, and returns the median result among the c sets. It can be shown that this brings down p_f by a factor of $1/C$ with $c = O(\log C)$. In

Algorithm 2: Weigted-One-Sided-Pair-Sampling(G)

1 Sample a vertex $u \in L$ with probability d_u/m, and a vertex $v \in L$ with probability d_v/m;
2 **if** $u == v$ **then** return 0;
3 $\bowtie_{u,v} \leftarrow \binom{\xi_{u,v}}{2}$;
4 **return** $\frac{m^2}{2d_u d_v} \cdot \bowtie_{u,v}$;

practice, p_f is usually a pre-defined constant, e.g., 1/32 in [55], from which we obtain the values of c and t accordingly. We follow their setting of p_f.

3.2 Weighted One-Sided Pair Sampling

Recall from Sec. 3.1 that a main advantage of the pair sampling algorithm is that it captures a quadratic number of butterflies (i.e., $\bowtie_{u,v} = \binom{\xi_{u,v}}{2}$, for vertex pair $u, v \in L$) in linear time (by computing $\xi_{u,v}$), leading to significant cost savings. Observe that the amount of cost savings here is amplified by a large value of $\xi_{u,v}$, due to the quadratic relationship. This motivates us to sample pairs of vertices with a large common neighbor set with higher probabilities, so as to capture an even larger number of butterflies at once.

Yet, the size of the intersection set $\xi_{u,v}$ is unknown in advance, and pre-computing their values is clearly infeasible, which takes $\Omega(|L|^2)$ time and space. Hence, we cannot sample a pair u, v with probability based precisely on the unknown $\xi_{u,v}$. As a heuristic, we instead sample u and v independently, with probabilities proportional to $d_u = |N[u]|$ and $d_v = |N[v]|$, respectively. This leads to our *weighted one-sided pair sampling* algorithm, shown in Alg. 2. Specifically, the algorithm assigns a sampling probability of $\frac{d_u}{m}$ to each vertex $u \in L$ (Line 1), where m is the number of edges in G. Note that since G is bipartite, the sum of all vertices degrees in L equals m, meaning that the sampling probabilities of all vertices in L sum up to 1. Then, the algorithm computes $\bowtie_{u,v}$ in Lines 2-3. By multiplying a factor of $m^2/(2d_u d_v)$, an unbiased estimation can be obtained (Line 4). Next, we explain why this estimator is unbiased.

Let us focus on scaling the result to obtain an unbiased estimate of $\bowtie$ for the whole graph G. Similar to the analysis of the pair sampling algorithm in the previous subsection, we use X_i to indicate whether the i-th butterfly contains u and v. Specifically, $X_i = \frac{1}{d_u d_v}$ when the butterfly contains both vertices u and v, and 0 otherwise. Since u, v are sampled with probability proportional to their degrees, we derive $\mathbb{E}[X_i] = \frac{1}{d_u d_v} \cdot \frac{2d_u d_v}{m^2} = \frac{2}{m^2}$. Define random variable $X = \sum_{i=1}^{\bowtie} X_i$. Clearly, $\mathbb{E}[X] = \sum_{i=1}^{\bowtie} \mathbb{E}[X_i] = 2 \cdot \bowtie / m^2$. Let Y_{WP} be the random variable of the result of Alg. 2. Then, we have $Y_{WP} = X \cdot m^2/2$, which leads to the following lemma.

Lemma 3.4. $\mathbb{E}[Y_{WP}] = \bowtie$.

Example 3.5. Consider the bipartite graph G in Fig. 2. Suppose we sample vertices from $L = \{u_1, u_2, u_3, u_4, u_5\}$, where the degree of each vertex is L is $\{4, 3, 4, 2, 2\}$, respectively. Assume that we sampled u_2 with a probability of 3/15 and another vertex u_3 with a probability of 4/15. Then we calculate $\bowtie_{u_1,u_2} = 1$. By multiplying a factor of $15^2/24$, we get an estimation of $\bowtie$ which is 9.375. We can get a more accurate estimation result by sampling multiple times.

The variance of Y_{WP} can be derived in a similar way as that of Y_P in the previous subsection. In particular, given the i-th butterfly, let $d_u^{(i)} d_v^{(i)}$ be the product of the degrees of its two vertices in L (note that a butterfly always contains exactly two vertices from L). Let $SP_{2,L}$ be the set of butterfly pairs that share the same two vertices in L (note that $SP_{2,L}$ is a set rather than a count, as is the

case for $P_{2,L}$ in the previous subsection). Given a butterfly pair p, we define $d_u^{(p)} d_v^{(p)}$ as the product of the degrees of the shared two vertices on the left side of p. Then, we derive an upper bound for the variance of Y_{WP}, as follows.

Lemma 3.6. $\mathbb{V}\mathrm{ar}[Y_{WP}] \leq \frac{m^2}{2}\left(\sum_{i=1}^{\boxtimes} \frac{1}{d_u^{(i)} d_v^{(i)}} + \sum_{p \in SP_{2,L}} \frac{1}{d_u^{(p)} d_v^{(p)}}\right)$.

Proof. $\mathbb{V}\mathrm{ar}[Y_{WP}]$ can be written as follows:

$$\left(\frac{m^4}{4}\right)\mathbb{V}\mathrm{ar}[\sum_{i=1}^{\boxtimes} X_i] = \left(\frac{m^4}{4}\right)\left(\frac{2}{m^2}\left(\sum_{i=1}^{\boxtimes}\left(\frac{1}{d_u^{(i)} d_v^{(i)}} - \frac{2}{m^2}\right)\right) + \sum_{i \neq j}\left(\mathbb{E}[X_i X_j] - \mathbb{E}[X_i]\mathbb{E}[X_j]\right)\right).$$

In the last equality, u, v are the left vertices of i-th butterfly. Let's still consider the butterfly pair (i, j): if butterflies i and j share less than 2 vertices, it is impossible for $\boxtimes_{u,v}$ to count butterfly i and j at the same time. In this case, $\mathbb{E}[X_i X_j] = 0, \mathbb{C}\mathrm{ov}\left(X_i, X_j\right) = -4/m^4$. Next, we consider that butterflies i and j both contain u, v, in which case the vertices on R may be three or four. In both cases, their covariances are the same, so we consider them together. We have:

$$\mathbb{E}[X_i X_j] = \frac{2 d_u^{(i)} d_v^{(i)}}{m^2} \cdot \left(\frac{1}{d_u^{(i)} d_v^{(i)}}\right)^2, \mathbb{C}\mathrm{ov}\left(X_i, X_j\right) = \frac{2}{m^2 d_u^{(i)} d_v^{(i)}} - \frac{4}{m^4}.$$

Let $SP_{2,L}$ denote the set of butterfly pairs that share the same two vertices in side L. Then, $\mathbb{V}\mathrm{ar}[Y_{WP}]$ can be written as:

$$\begin{aligned}
&\frac{m^4}{4}\left(\frac{2}{m^2}\sum_{i=1}^{\boxtimes}\left(\frac{1}{d_u^{(i)} d_v^{(i)}} - \frac{2}{m^2}\right) + \sum_{p \in SP_{2,L}} \frac{2}{m^2 d_u^{(p)} d_v^{(p)}} - \binom{\boxtimes}{2} \cdot \frac{4}{m^4}\right) \\
&= \frac{m^2}{2}\left(\sum_{i=1}^{\boxtimes} \frac{1}{d_u^{(i)} d_v^{(i)}} + \sum_{p \in SP_{2,L}} \frac{1}{d_u^{(p)} d_v^{(p)}}\right) - \boxtimes - \binom{\boxtimes}{2} \leq \frac{m^2}{2}\left(\sum_{i=1}^{\boxtimes} \frac{1}{d_u^{(i)} d_v^{(i)}} + \sum_{p \in SP_{2,L}} \frac{1}{d_u^{(p)} d_v^{(p)}}\right),
\end{aligned}$$

which finishes our proof. □

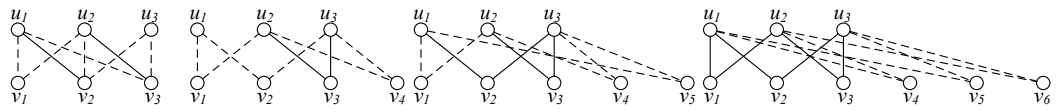
To simplify our notations, we define $t_1 = \sum_{i=1}^{\boxtimes} \frac{1}{d_u^{(i)} d_v^{(i)}}$ and $t_2 = \sum_{p \in SP_{2,L}} \frac{1}{d_u^{(p)} d_v^{(p)}}$. Let $\bar{Y}_{WP}$ be the average of t independent calculation of Y_{WP}. By Chebyshev's inequality, we have

$$\Pr[|\bar{Y}_{WP} - \boxtimes| \geq \epsilon \boxtimes] \leq \frac{\mathbb{V}\mathrm{ar}[\bar{Y}_{WP}]}{\epsilon^2 \boxtimes^2} \leq \frac{m^2(t_1 + t_2)}{2t\epsilon^2 \boxtimes^2} = p_f.$$

This leads to the following lemma. Similar to the case of pair sampling described in the previous subsection, p_f can be reduced via the median trick. We omit further details for brevity.

Lemma 3.7. *With $t = O\left(m^2(t_1 + t_2)/\boxtimes^2\right)$, we obtain an ϵ-error estimation for the number of butterflies in $O\left(m(t_1 + t_2) d_L^2/\boxtimes^2\right)$ expected time with $1 - p_f$ probability, where $d_L^2 = \sum_{i \in L} d_i^2$.*

Proof. Firstly, assume L is the sampled side of the graph. Sampling two vertices can be conducted by using the alias method [71] with $O(1)$ sampling time or BUS structure [85] in dynamic scenario. Note that since the probability of a vertex being sampled is greater with a larger degree, the expected time for a single run of the algorithm is $O\left(\sum_{v \in L} \frac{d_v}{m} \cdot d_v\right) = O\left(\frac{d_L^2}{m}\right)$, where $d_L^2 = \sum_{i \in L} d_i^2$. Since we sample $t = O\left(\frac{m^2(t_1+t_2)}{p_f \epsilon^2 \boxtimes^2}\right)$ times, the total expected time complexity is $O\left(m\left(t_1 + t_2\right) d_L^2/\boxtimes^2\right)$. □



(a) Type 1 (b) Type 2 (c) Type 3 (d) Type 4

Fig. 4. Four types of bi-triangle pairs

Notice that the alias sampling structure [71] can be constructed in $O(|L|)$ time when loading the input graph. The following theorem states that under a power-law random bipartite graph model, to be detailed in Sec. 6.1, our main proposal, weighted pair sampling, is expected to be more efficient than both the current state-of-the-art FastEdge [55] and the one-sided pair sampling method described in Sec. 3.1, under comparable accuracy and failure probability levels.

Theorem 3.8. *Under the power-law random bipartite graph model (refer to Sec. 6.1), with the same error bound and failure probability for approximate butterfly counting, the ratio between the total running time of FastEdge and weighted pair sampling is $O(m/n)$ with high probability; the ratio between the running time of pair sampling and weighted pair sampling is $O\left(|L|\Delta^2/d_L^2\right)$ with high probability.*

All omitted proofs can be found in our technical report [2]. According to the above theorem, since typically $m > n$ and $d_L^2 = \sum_{v \in L} d_v^2 < |L|\Delta^2$ (d_L^2 is defined in Lem. 3.7, Δ is the maximum degree of L in the power-law random bipartite graph model), weighted pair sampling is expected to outperform both FastEdge and pair sampling. The detailed derivations can be found in Sec. 6.

4 APPROXIMATE BI-TRIANGLE COUNTING

4.1 Baseline Methods

Although there is no known work that specifically studies the problem of approximate bi-triangle counting, it is possible to combine local sampling schemes designed for approximate butterfly counting with exact solutions for bi-triangle counting, both described in Sec. 2.2. Thus, we establish a few baseline solutions with such combinations. The local sampling methods overviewed in Sec. 2.2 include vertex sampling, edge sampling, and wedge sampling. Unlike the case for butterfly counting, however, computing the number of bi-triangles is rather expensive given a sampled vertex, edge, or wedge. This issue can be mitigated via the specialized V-LCount (resp. E-LCount) algorithm [84] for exact bi-triangle counting with a given vertex (resp. edge), with simple adaptations. For wedge sampling, although neither V-LCount nor E-LCount applies, we can design an algorithm by adapting WJ-Count [84] for this purpose.

Even with the help of these specialized counting algorithms, local sampling still incurs considerable computational overhead, especially with a large number of samples necessary for obtaining high accuracy, as shown later in our experiments in Sec. 7. Similar to the case of butterfly counting, the root cause is that the local sampling solutions of previous work [55] are direct adaptations of methods designed for general, unipartite graphs, which fail to exploit the special properties of bipartite graphs. Hence, we propose to extend our weighted pair sampling to bi-triangle counting, which leads to a weighted triple sampling algorithm, presented next. Of course, it is also natural to extend our unweighted pair sampling method to the unweighted triple sampling method. We will compare such an extension as well in our experiment. For the interest of space, we omit the discussion of the unweighted case.

4.2 Weighted One-Sided Triple Sampling

Similar to weighted pair sampling described in Sec. 3.2, we sample vertices from one side (say, L) of the input bipartite graph G, with probabilities proportional to their degrees. Here, a bi-triangle involves three vertices on each side; hence, we sample three vertices $u, v, w \in L$. The neighbor lists of these vertices form an intersection set $N[u] \cap N[v] \cap N[w]$, as well as 3 pair-wise intersections: $N[u] \cap N[v]$,$N[u] \cap N[w]$, $N[v] \cap N[w]$. Recap that $\xi_{u,v} = |N[u] \cap N[v]|$ and $\xi_{u,v,w} = |N[u] \cap N[v] \cap N[w]|$. The number of bi-triangles $\bowtie_{u,v,w}$ containing u, v, w can be computed through the cardinalities of these intersection sets, as follows.

Lemma 4.1 ([84]). *Given three vertices $u, v, w \in L$, the number of bi-triangles containing these vertices $\bowtie_{u,v,w}$ can be computed by: $\xi_{u,v} \cdot \xi_{v,w} \cdot \xi_{w,u} - (\xi_{u,v} \cdot \xi_{v,w} + \xi_{u,v} \cdot \xi_{w,u} + \xi_{v,w} \cdot \xi_{w,u} - 2)\xi_{u,v,w}$.*

Alg. 3 shows the pseudo-code for weighted triple sampling, which performs sampling in Line 1 and local counting in Lines 2-3. We focus on the scaling step in Line 4, which obtains an unbiased estimate for the bi-triangle count $\bowtie$ in G. Assume that the $\bowtie$ bi-triangles are numbered from 1 to $\bowtie$. We define X_i such that $X_i = \frac{1}{d_u d_v d_w}$ when the i-th butterfly contains u, v, w, and $X_i = 0$ otherwise. Then, we derive that $\mathbb{E}[X_i] = \frac{1}{d_u d_v d_w} \cdot \frac{(3!) d_u d_v d_w}{m^3} = \frac{6}{m^3}$. Let $X = \bowtie_{u,v,w} = \sum_{i=1}^{\bowtie} X_i$ be the sum of $\bowtie$ random variables. We have $\mathbb{E}[X] = \sum_{i=1}^{\bowtie} \mathbb{E}[X_i] = 6 \cdot \frac{\bowtie}{m^3}$. Let Y_{WT} denote the random variable corresponding to the result of Alg. 3. Clearly, $Y_{WT} = X \cdot \frac{m^3}{6}$, which leads to the following result.

Lemma 4.2. $\mathbb{E}[Y_{WT}] = \bowtie$.

The variance of Y_{WT} can be derived by the sum of X_i. Let $d_u^{(i)} d_v^{(i)} d_w^{(i)}$ be the product of the degrees of the three vertices in L for the i-th bi-triangle. Let $SP_{3,L}$ be the set of bi-triangle pairs that share the same three vertices in L. Fig. 4 shows examples of bi-triangle pairs that share three vertices in $L = \{u_1, u_2, u_3\}$; in all 4 examples, one bi-triangle is u_1-v_1-u_2-v_2-u_3-v_3-u_1, and the other is shown in dashed lines. Given a pair $p \in SP_{3,L}$, we define $d_u^{(p)} d_v^{(p)} d_w^{(p)}$ be the product of the degrees of the three vertices in L shared by the pair of bi-triangles in p. Let $t_3 = \sum_{i=1}^{\bowtie} 1/(d_u^{(i)} d_v^{(i)} d_w^{(i)})$ and $t_4 = \sum_{p \in SP_{3,L}} 1/(d_u^{(p)} d_v^{(p)} d_w^{(p)})$, we have the following lemma.

Lemma 4.3. $\mathbb{V}\mathrm{ar}[Y_{WT}] \leq \frac{m^3}{6}(t_3 + t_4)$.

Proof. We analyze the variance of Y_{WT} by analyzing the pair composed of different bi-triangles. $\mathbb{V}\mathrm{ar}[Y_{WT}]$ can be written as follows:

$$\mathbb{V}\mathrm{ar}[\frac{m^3}{6}\sum_{i=1}^{\bowtie} X_i] = \left(\frac{m^6}{36}\right)\mathbb{V}\mathrm{ar}[\sum_{i=1}^{\bowtie} X_i] = \left(\frac{m^6}{36}\right)\left(\frac{6}{m^3}\left(\sum_{i=1}^{\bowtie}\left(\frac{1}{d_u^{(i)} d_v^{(i)} d_w^{(i)}} - \frac{6}{m^3}\right)\right) + \sum_{i \neq j} \mathbb{C}\mathrm{ov}\left(X_i, X_j\right)\right).$$

For the last equation, u, v, w are left-side vertices of the i-th bi-triangle. Consider the bi-triangle pair (i, j): If bi-triangles i and j share less than three vertices on L, it is impossible for $\bowtie_{u,v,w}$ to compute bi-triangle i and j at the same time. In this case, $\mathbb{E}[X_i X_j] = 0, \mathbb{C}\mathrm{ov}\left(X_i, X_j\right) = -\frac{36}{m^6}$. If bi-triangles i and j contain u, v, w, in which case the vertices on the other side may be three to six. In both cases, the covariances are the same. So we consider them together. The expectation of the product of X_i and X_j is as follows.

$$\mathbb{E}[X_i X_j] = \frac{6 d_u^{(i)} d_v^{(i)} d_w^{(i)}}{m^3} \cdot \left(\frac{1}{d_u^{(i)} d_v^{(i)} d_w^{(i)}}\right)^2 = \frac{6}{m^3 d_u^{(i)} d_v^{(i)} d_w^{(i)}}.$$

We have the following result for the covariance of X_i and X_j.

$$\mathbb{C}\mathrm{ov}\left(X_i, X_j\right) = \mathbb{E}[X_i X_j] - \mathbb{E}[X_i]\mathbb{E}[X_j] = \frac{6}{m^3 d_u^{(i)} d_v^{(i)} d_w^{(i)}} - \frac{36}{m^6}.$$

Algorithm 3: Weighted-One-Sided-Triple-Sampling(G)

1 Sample vertices $u, v, w \in L$ with probabilities $\frac{d_u}{m}$, $\frac{d_v}{m}$, $\frac{d_w}{m}$, respectively;
2 **if** $u == v$ *or* $v == w$ *or* $u == w$ **then** return 0;
3 Compute $\bowtie_{u,v,w}$ according to Lemma 4.1;
4 **return** $\frac{m^3}{6 d_u d_v d_w} \cdot \bowtie_{u,v,w}$;

Then let $SP_{3,L}$ denote the set of bi-triangle pairs that share the same three vertices in side L. The $\mathbb{V}\mathrm{ar}[Y_{WT}]$ can be written as:

$$\frac{m^3}{6}\left(\sum_{i=1}^{\bowtie} \frac{1}{d_u^{(i)} d_v^{(i)} d_w^{(i)}} + \sum_{p \in SP_{3,L}} \frac{1}{d_u^{(p)} d_v^{(p)} d_w^{(p)}}\right) - \bowtie - \binom{\bowtie}{2} \leq \frac{m^3}{6}\left(\sum_{i=1}^{\bowtie} \frac{1}{d_u^{(i)} d_v^{(i)} d_w^{(i)}} + \sum_{p \in SP_{3,L}} \frac{1}{d_u^{(p)} d_v^{(p)} d_w^{(p)}}\right).$$

This finishes the proof. □

Let $\bar{Y}_{WT}$ be the average of t independent calculation of Y_{WT}. By Chebyshev's inequality, we have

$$Pr[|\bar{Y}_{WT} - \bowtie| \geq \epsilon \bowtie] \leq \frac{\mathbb{V}\mathrm{ar}[\bar{Y}_{WT}]}{\epsilon^2 \bowtie^2} \leq \frac{m^3(t_3 + t_4)}{6t\epsilon^2 \bowtie^2} = p_f.$$

Hence, we arrive at the following lemma.

Lemma 4.4. *With $t = O\left(m^3(t_3 + t_4)/\bowtie^2\right)$, we obtain an ϵ-error estimation for the number of butterflies in $O\left(m^2(t_3 + t_4)d_L^2/\bowtie^2\right)$ expected time with $1 - p_f$ probability, where $d_L^2 = \sum_{i \in L} d_i^2$.*

Proof. By choosing L as the sampled side, the operation of sampling three vertices can be conducted by using the alias method. The expected time for a single run of the algorithm is $O\left(\sum_{v \in L} \frac{d_v}{m} \cdot d_v\right) = O\left(\frac{d_L^2}{m}\right)$. Since we need sample $t = O\left(\frac{m^3(t_3+t_4)}{p_f \epsilon^2 \bowtie^2}\right)$ times, the total expected time complexity is $O\left(\frac{m^2(t_3+t_4)d_L^2}{\bowtie^2}\right)$. □

The following theorem establishes the advantage of weighted triple sampling compared to triple sampling, under the random graph model that is clarified in Sec. 6.

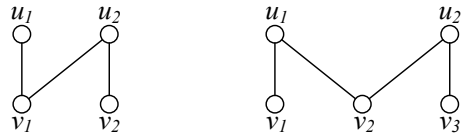
Theorem 4.5. *Under the power-law random bipartite graph model in Sec. 6.1, with the same error bound and failure probability parameters for approximate bi-triangle counting, the ratio between the expected running time of triple sampling and weighted triple sampling is $O\left(|L|^2\Delta^3/d_L^2 m\right)$ with high probability.*

It is easy to verify that $m < |L|\Delta$ and $d_L^2 = \sum_{v \in L} d_v^2 < |L|\Delta^2$. Therefore, weighted triple sampling is expected to be faster than triple sampling to reach comparable error levels.

5 APPROXIMATE CLUSTERING COEFFICIENT

Next, we show how to integrate our approximate butterfly counting and bi-triangle counting algorithms for efficient estimation of clustering coefficients with high quality.

Butterfly-based clustering coefficient. The butterfly-based clustering coefficient $C_{\boxtimes}$ is defined as $4 \times \boxtimes_G / \boxslash_G$, where $\boxtimes_G$ denotes the number of butterflies in G, and $\boxslash_G$ represents the number of 3-paths in G. We define $\boxslash_{(u,v)}$ as the number of 3-paths with a central edge u, v. It is important to note that the central edge of a given 3-path is unique. For instance, in Fig. 5(a), the 3-path (u_1, v_1, u_2, v_2) has a central edge (v_1, u_2). The number of 3-paths in which central edge is (v_1, u_2) can be easily obtained by multiplying the degrees of u_2 and v_1. To compute the total number of



(a) Example 3-path (b) Example 4-path

Fig. 5. Bipartite graph motifs: 3-path and 4-path

3-paths in G, one can enumerate each edge (u, v) and accumulate the number of 3-paths with (u, v) as the central edge. Yet, this approach still involves traversing the entire graph. Hence, we propose using sampling to estimate the number of 3-paths. Given the simple structure of 3-paths, we employ edge sampling to estimate their count. One may use other local samplings to estimate $\bowtie_G$. As 3-path is relatively simple and edge sampling is effective enough, we do not discuss other estimators. In edge sampling, the sampling unit S is an edge $u, v \in E$. The overall estimation process consists of uniformly and randomly sampling an edge (u, v) from E, then computing the number of 3-paths $\bowtie_{(u,v)}$ for which u, v is the central edge. By multiplying the local count $\bowtie_{(u,v)}$ with $m/3$, we obtain an unbiased estimator for $\bowtie_G$. We then take the average over multiple samples to reduce the variance. Then, the approximate butterfly-based clustering coefficient $\widetilde{C}_{\boxtimes}$ can be estimated as $\widetilde{C}_{\boxtimes} = 4 \times \widetilde{\boxtimes}_G / \widetilde{\bowtie}_G$, where $\widetilde{\boxtimes}_G$ and $\widetilde{\bowtie}_G$ denote the approximate number of butterflies and 3-paths in G, respectively. $\widetilde{\boxtimes}_G$ can be estimated using Alg. 2. In practice, we could allocate equal elapsed time to both butterfly and 3-path estimation, thereby obtaining estimates for both the numerator and the denominator. In Sec. 7, we evaluate the performance of approximating the butterfly-based clustering coefficient, where 3-paths are estimated using edge sampling, and the number of butterflies is approximated through various local sampling methods.

Bi-triangle-based clustering coefficient. For the calculation of bi-triangle-based clustering coefficient, we already have an efficient algorithm (Algorithm 3) to approximate the number of bi-triangles. Based on previous work [84] for the calculation of the bi-triangle-based clustering coefficient, the bottleneck now is how to quickly count the number of 4-paths centered on L or R. Since the algorithm for calculating the exact number of 4-paths will take $O(|V| + \alpha|E|)$ time [15], where α is the arboricity of G. We also developed the corresponding local sampling method to approximate the number of 4-paths. As mentioned in Sec. 1, the bi-triangle-based clustering coefficient is particularly concerned with the cohesiveness of a one-sided vertex set. Assume we want to measure the clustering coefficient of R. In this case, we can apply the one-sided weighted pair sampling by sampling two vertices u, v from L. In this case, u, v will correspond to (u_1, u_2) in Fig. 5(b), and the probability of each vertex being sampled is related to its degree. The number of 4-paths containing u, v can be calculated by $\xi_{u,v} \cdot \left[(d_u - 1) \cdot (d_v - 1) - (\xi_{u,v} - 1)\right]$, where $\xi_{u,v}$ is the number of common neighbors of u and v. We can obtain an unbiased estimator for the 4-paths by using a factor similar to the weighted one-sided pair sampling for butterfly counting to adjust.

6 THEORETICAL ANALYSIS

Sec. 6.1 presents a power-law random bipartite graph model for the analysis, and Sec. 6.2 presents the proofs of previous sections.

6.1 Power-Law Random Bipartite Graph Model

Previous research [70] shows that many real bipartite graphs follow the power-law distribution, similar to the case of unipartite graphs (commonly known as scale-free graphs). Thus, we adapt the power-law random graph model [4] to bipartite graphs to analyze our algorithms, which allows us to obtain a fine-grained estimate for certain values that depend upon the graph structure, e.g.,

$P_{2,L}$ introduced in Lem. 3.2. With these, we derive more precise bounds on the running time of our algorithms and that of our competitors.

Given a power-law bipartite graph $G = \langle V = (L, R), E \rangle$, for any x, y such that y is the number of vertices with degree x in L, we have $\log y = \alpha - \beta \log x$, i.e., $|\{v : d_v = x, v \in L\}| = y = \frac{e^\alpha}{x^\beta}$. To simplify the analysis, we only assume that one side L follows the power law. Usually, $0 < \beta < 2$, and α is the logarithm of the size of the graph. Then, we can construct a power-law random bipartite graph. First, we generate a degree sequence for vertices in L with the above degree distribution. Next, we randomly connect vertices in R based on the degree of each vertex. We let $\Delta^2 > R$ be used to subsequently simplify our analysis, where Δ is the maximum degree of L. Almost all data from real graphs satisfy this point, so this assumption is natural. Accordingly, we have

$$|L| = \sum_{x=1}^{\Delta} \frac{e^\alpha}{x^\beta} = \zeta(\beta, \Delta)\, e^\alpha, |E| = \sum_{x=1}^{\Delta} \frac{e^\alpha}{x^\beta} \cdot x = \zeta(\beta - 1, \Delta)\, e^\alpha,$$

where ζ is a bivariate function: $\zeta(x, y) = \sum_{i=1}^{y} \frac{1}{i^x}$. In the following, we omit Δ and use $\zeta(\beta)$ to denote $\zeta(\beta, \Delta)$ if the context is clear.

Lemma 6.1. *The ζ function has the following useful properties:*

$$\begin{aligned}
\zeta(\beta, \Delta) &\le c_\beta && \beta > 1;\\
\zeta(\beta, \Delta) &\to \ln \Delta && \beta = 1;\\
\frac{(\Delta+1)^{1-\beta} - 1}{1-\beta} \le &\zeta(\beta, \Delta) \le \frac{\Delta^{1-\beta}}{1-\beta} && 0 < \beta < 1;\\
\frac{\Delta^{1-\beta}}{1-\beta} \le &\zeta(\beta, \Delta) \le \frac{(\Delta+1)^{1-\beta} - 1}{1-\beta} && \beta \le 0.
\end{aligned}$$

6.2 Proofs

Due to limited space, we present a proof sketch of the theorems. Interested readers are referred to our technical report [2] for details.

Proof sketch of Thm. 3.8. Let's start by analyzing the variance of pair sampling. According to Lem. 3.2, $\mathbb{V}\mathrm{ar}[Y_P] \le \binom{|L|}{2}\left(\boxtimes + P_{2,L}\right)$. Since in the random graph model, vertices in L and those in R are connected randomly. The number of butterflies containing $u, v \in L$ is derived by common neighbors of u, v. For the number of common neighbors of u and v, clearly, $\mu = \mathbb{E}[\xi_{u,v}] = d_u \cdot d_v / |R|$. We can find it follows a sampling process without replacement. Then we can use Hoeffding's inequality [7] (Equation 1) to bound the common neighbors of u and v with high probability. Let X be the random variable of the common neighbors of u, v. For all $\epsilon > 0$, we have

$$\Pr[X - \mu \ge \epsilon \cdot d_v] \le exp\left(-2\epsilon^2 \cdot d_v\right). \tag{1}$$

Let ϵ be $c \cdot d_u / R$ with $c > 0$ and take a large enough value for c. It allows the common neighbors of u, v to be $O(d_u d_v / |R|)$ with high probability. Next, we use union bound so that the common neighbors of any pair of vertices in L are bounded in the corresponding upper value with high probability. The following analyses are based on events occurring with high probability:

$$\boxtimes_{u,v} = \binom{\xi_{u,v}}{2} = O\left(\frac{d_u^2 d_v^2}{|R|^2}\right).$$

Analogously, we can perform a theoretical analysis of the terms $\boxtimes, P_{2,L}$ and obtain the expressions of the variance of different sampling methods. Detailed analysis can be found in the technical report [2]. Then, we can compare the running time of different methods with the same parameters for

Table 2. Summary of datasets

Dataset	type	$\|L\|$	$\|R\|$	$\|E\|$	wedges	butterflies	bi-triangles
Wiki-de	authorship	1.03×10^{06}	5.81×10^{06}	5.52×10^{07}	1.81×10^{12}	1.64×10^{12}	2.30×10^{18}
Reuters	text	7.81×10^{05}	2.84×10^{05}	6.06×10^{07}	1.55×10^{12}	7.49×10^{12}	2.47×10^{19}
Delicious (Deli)	interaction	8.33×10^{05}	4.51×10^{06}	8.20×10^{07}	1.45×10^{12}	2.67×10^{13}	1.76×10^{20}
Gottron	text	5.52×10^{05}	1.17×10^{06}	8.36×10^{07}	1.60×10^{12}	1.94×10^{13}	1.07×10^{20}
LiveJournal (LJ)	affiliation	3.20×10^{06}	7.49×10^{06}	1.12×10^{08}	2.70×10^{12}	3.30×10^{12}	6.11×10^{18}
Wiki-en	authorship	8.12×10^{06}	4.25×10^{07}	2.56×10^{08}	3.37×10^{13}	2.13×10^{13}	1.03×10^{20}
Yahoo	rating	1.00×10^{06}	6.25×10^{05}	2.57×10^{08}	4.63×10^{12}	1.01×10^{14}	1.01×10^{21}
Orkut	affiliation	2.78×10^{06}	8.73×10^{06}	3.27×10^{08}	2.53×10^{12}	2.21×10^{13}	1.28×10^{20}

error guarantees and failure probabilities. Following the previous work [55], we set p_f to 1/32 and ϵ to the same value for all algorithms. Further, the ratio of running times costs for edge sampling, pair sampling, and weighted pair sampling with the same parameters can be calculated as:

$$\mathbb{V}\mathrm{ar}[Y_E]\frac{m\Delta}{n} : \mathbb{V}\mathrm{ar}[Y_P]\frac{m}{|L|} : \mathbb{V}\mathrm{ar}[Y_{WP}]\frac{d_L^2}{m}.$$

Omit the equation derivation, the ratio between edge sampling and weighted pair sampling is

$$\mathbb{V}\mathrm{ar}[Y_E]\frac{m\Delta}{n} : \mathbb{V}\mathrm{ar}[Y_{WP}]\frac{d_L^2}{m} = O\left(\frac{m}{n}\right).$$

The ratio between edge sampling and weighted pair sampling is

$$\mathbb{V}\mathrm{ar}[Y_P]\frac{m}{|L|} : \mathbb{V}\mathrm{ar}[Y_{WP}]\frac{d_L^2}{m} = O\left(\frac{|L|\Delta^2}{d_L^2}\right).$$

As $m > n, d_L^2 = \sum_{v \in L} d_v^2 \leq |L| \cdot \Delta^2$, we derive that weighted pair sampling should be better than edge sampling and pair sampling. □

7 EXPERIMENTS

We have implemented all proposed algorithms in C++ and conducted an extensive experimental study comparing their performance in various aspects. The implementations of the competitors are obtained from their respective authors, and we use hyperparameter settings consistent with those in their respective papers. Our code is available at [2]. All experiments are conducted on a Linux machine with an Intel Xeon CPU @ 2.30GHz and 384GB memory.

7.1 Experimental Settings

Datasets. In the experiments, we use eight real large bipartite networks (available in KONECT [1]), commonly used in previous work [55, 84]. Among these, Reuters and Gottron are text networks; Wiki-de and Wiki-en are authorship networks; LiveJournal and Orkut are affiliation networks; Deli is an interaction network; Yahoo is a rating network. Following previous works [55, 84], we perform the following preprocessing steps for each dataset: removing self-loops and duplicate edges, converting to an undirected graph, relabeling vertex IDs to 1, 2, ..., n. Tab. 2 shows statistics on these datasets.

Algorithms. We include the following algorithms in our experimental comparisons: (a) Pair Sampling (Algorithm 1), Weighted Pair Sampling (Algorithm 2), and the current state-of-the-art algorithm FastEdge [55] for approximate butterfly counting, which outperforms all other methods of previous work [55] according to their experiments. (b) The local sampling algorithms for approximate bi-triangle counting, described in Sec. 4, which include Vertex Sampling, Edge

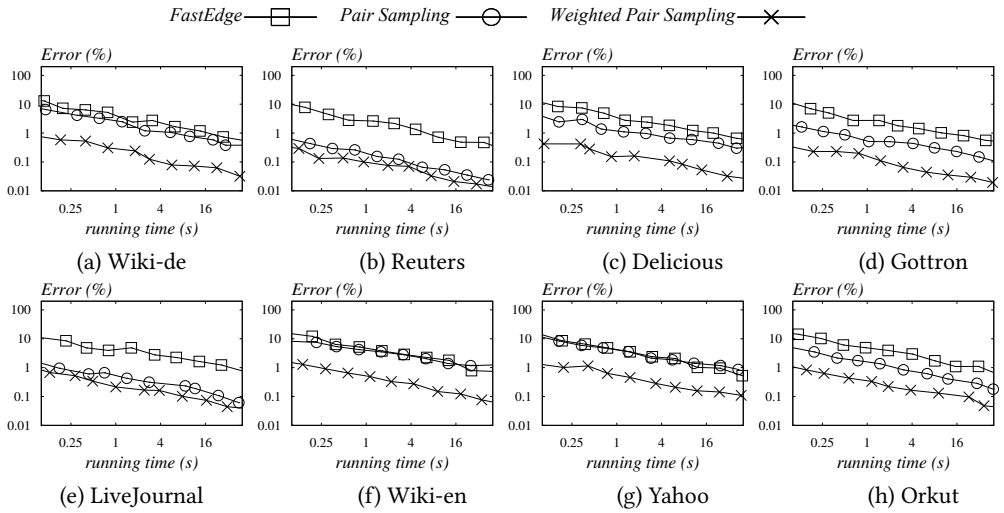


Fig. 6. Relative error vs running time, for approximate butterfly counting

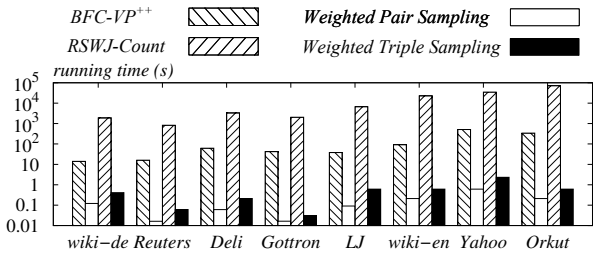
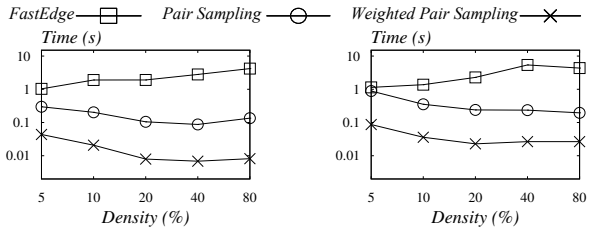


Fig. 7. Time to calculate the number of motifs: exact algorithm and approximate algorithm to obtain 1% relative error.

Sampling, Wedge Sampling, Triple Sampling, and Weighted Triple Sampling. (c) The state-of-the-art exact butterfly counting algorithm (BFC-VP^{++}) [76] and the state-of-the-art exact bi-triangle counting algorithm (RSWJ-Count) [84]. The implementations of FastEdge, BFC-VP^{++}, and RSWJ-Count are provided by their respective inventors and are all implemented in C++. Following FastEdge [55], we exclude graph loading time in the experimental results. Note that FastEdge additionally incurs a high preprocessing cost as it needs to compute the sum of degrees of all neighbors of each vertex in V. In contrast, as mentioned in Sec. 3.2, our method only involves building an alias structure [71] for one-sided vertex set for weighted sampling, which can be done when loading the graph. In our experiment, we omit the preprocessing time of FastEdge and only report their sampling performance for comparison.

Evaluation Metrics and Parameters. Following previous work [55], for local sampling methods, each reported relative error is computed as the mean value of the results of multiple independent runs of its corresponding algorithm, each drawing one sample. More runs lead to lower error but increase the total running time. Thus, the result is presented as relative error over time. For each algorithm and each setting, we report the median results of $c = 50$ sets of experiments, each with multiple independent runs as described above. For all proposed algorithms with one-sided sampling, we choose the side corresponding to the smaller one of d_L^2, d_R^2 as the sampling side, where $d_L^2 = \sum_{v \in L} d_v^2$ and $d_R^2 = \sum_{v \in R} d_v^2$. To explain, in Thm, 3.8 and Thm. 4.5, it is clear that the sampling



(a) Gottron (b) Orkut

Fig. 8. Density vs running time to obtain 1% relative error, for approximate butterfly counting

complexity depends on d_L^2 if L is the sampling side. Hence, we choose the side with smaller value. We also want to emphasize that the choice of the side to be sampled can be efficiently decided during the graph loading process. In case the sampling is done on sub-graphs of the input graph, e.g., sub-graphs returned by different community search algorithms, this still can be done when outputting the sub-graphs.

7.2 Evaluation Results

Approximate butterfly counting. Fig. 6 plots the relative error (as a percentage) as a function of total running time for all local sampling algorithms for approximate butterfly counting, FastEdge, Pair Sampling, and Weighted Pair Sampling, on all datasets. The results demonstrate that the proposed Weighted Pair Sampling consistently achieves the best performance among all competitors, in all settings. In particular, with the same total sampling time, the error of Weighted Pair Sampling is usually at least one order of magnitude smaller than that of FastEdge. To reach the same error, the running time of Weighted Pair Sampling is usually two orders of magnitude smaller than that of FastEdge. This confirms the highly competitive effectiveness and efficiency of the proposed Weighted Pair Sampling approach. We observe that Pair Sampling does not always outperform FastEdge, e.g., the two show similar performance on datasets Wiki-de and Yahoo. Manual inspection reveals that the reason that Pair Sampling performs relatively poorly in these settings is exactly the motivation for Weighted Pair Sampling. In particular, in these datasets, a large number of butterflies contain high-degree vertices, which might be missed by Pair Sampling but are captured by Weighted Pair Sampling with high probability. Our main proposal, Weighted Pair Sampling, always demonstrates the best performance among all competitors. In Fig. 7, we also report the running time of the state-of-the-art exact butterfly counting algorithm BFC-VP^{++} as a comparison. We report the running time of the Weighted Pair Sampling method in Fig. 7 when it has a relative error of no more than 1% for each dataset. The results show that Weighted Pair Sampling achieves up to three orders of magnitude speed-up over the exact algorithm while still maintaining high accuracy with a relative error of less than 1%.

We further test the single-round running time and show the average variance for each local sampling algorithm. We observe that our Weighted Pair Sampling has a similar average variance as FastEdge, both of which are smaller than Pair Sampling. Yet, as we discussed in Sec. 3.2, the sampling time depends on both the variance and the single-round sampling time. Both Weighted Pair Sampling and Pair Sampling have far lower single-round running time than FastEdge, making them more efficient than FastEdge when achieving the same accuracy. Due to limited space, interested readers are referred to our technical report [2] for more details.

Impact of graph density. In this set of experiments, we examine the impact of graph density on all local sampling methods. Due to limited space, we focus on butterfly counting and compare the Weighted Pair Sampling, Pair Sampling, and FastEdge with varying graph density. We choose two representative datasets, Gottron and Orkut, with different scales of edges, and randomly reduce the

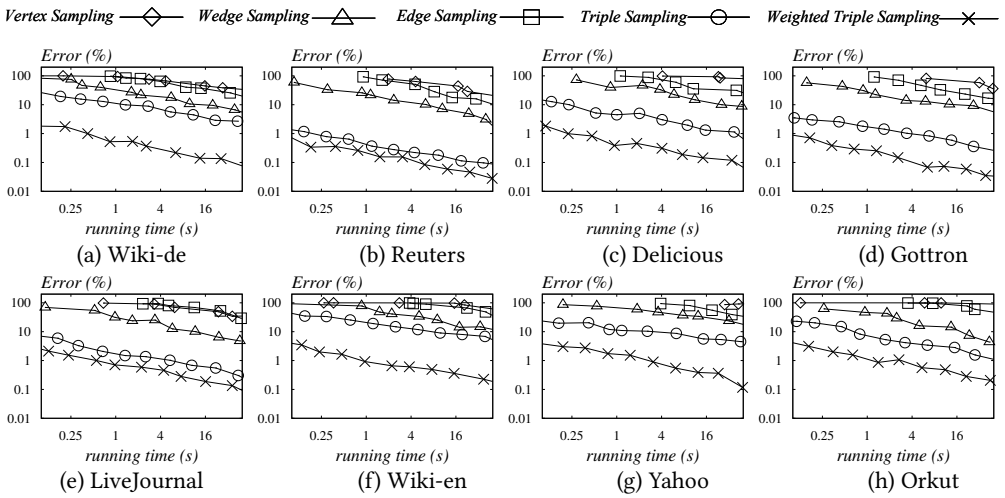


Fig. 9. Relative error vs. running time for approximate bi-triangle counting

graph density by keeping each edge with probability $p \in \{5\%, 10\%, 20\%, 40\%, 80\%\}$. Fig. 8 shows the sampling time by different algorithms to get 1% relative error on these two datasets with different densities. The results show that Weighted Pair Sampling consistently outperforms alternatives in different densities and is less sensitive to the graph density than Pair Sampling. This shows that our Weighted Pair Sampling is the favorable choice under different datasets with different densities.

Approximate bi-triangle counting. Fig. 9 shows the performance of different local sampling algorithms on all datasets, including Vertex Sampling, Edge Sampling, Wedge Sampling, Triple Sampling, and Weighted Triple Sampling, for approximate bi-triangle counting. Since bi-triangle is a more complex motif, local sampling algorithms for estimating bi-triangle counts generally require longer running times than those for approximate butterfly counting to reach a comparable error level. The results clearly demonstrate that the proposed Weighted Triple Sampling consistently and significantly outperforms all other methods. In Fig. 7, we also report the running time of the state-of-the-art exact bi-triangle counting algorithm RSWJ-Count as a comparison. We report the running time of the Weighted Triple Sampling method when it achieves a relative error less than 1%. The results highlight that Weighted Triple Sampling demonstrates a remarkable speed-up, up to four orders of magnitude, over the exact algorithm while ensuring high accuracy. We also test the single-round running time of each sampling method. Due to limited space, interested readers are referred to our technical report [2] for the details.

Approximate clustering coefficient. We use BFCC (resp. BTCC) to denote the butterfly-based (resp. bi-triangle-based) clustering coefficient. Fig. 10 shows the performance of the proposed BFCC estimation algorithm in Sec. 5 against alternatives. The experimental results show that our proposed solution significantly outperforms alternatives for estimating BTCC on all tested datasets when achieving the same accuracy. For the BTCC, we test the performance on the same side as that in approximate bi-triangle counting (Fig. 7). Fig. 11 shows the performance of our proposed solution for BTCC estimation against alternatives. This leads to similar conclusions as in the case of the approximate BFCC.

For the exact BFCC, we use BFC-VP^{++} to count the exact number of butterflies and spend $O(m)$ time to derive the exact number of 3-paths following [5]. For the exact BTCC, we use RSWJ-Count to derive the exact number of bi-triangles and count the exact number of 4-paths using the method of previous work [84], which takes $O(|V| + \alpha|E|)$ time. As we can see, the proposed framework can

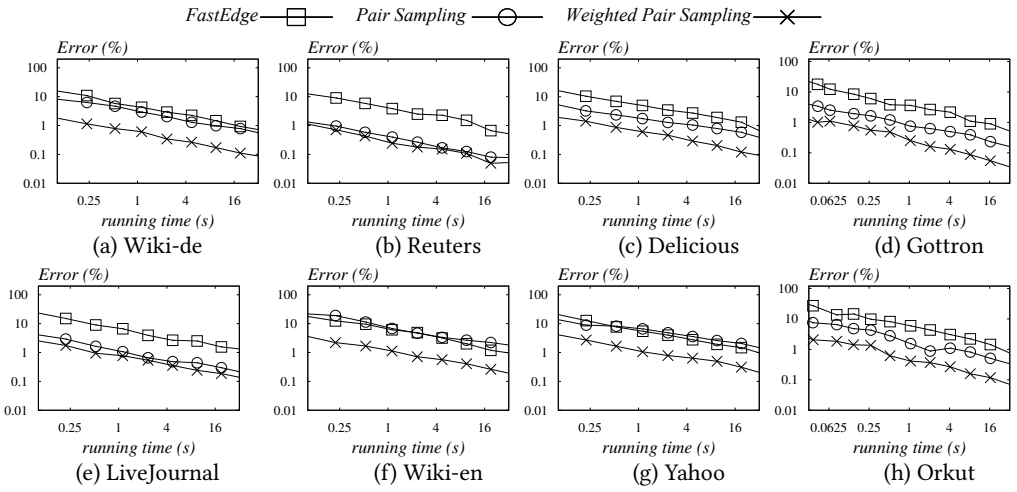


Fig. 10. Relative error vs. running time for BFCC

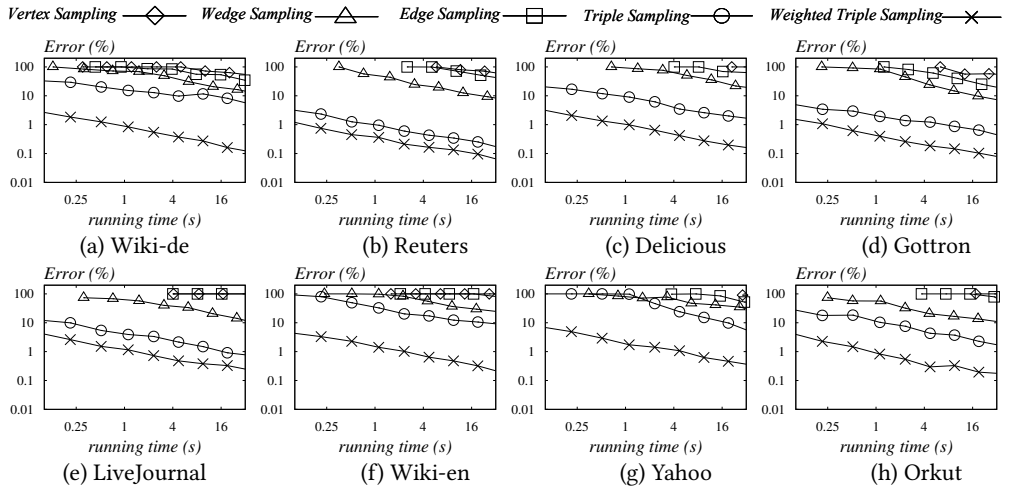


Fig. 11. Relative error vs. running time for BTCC

achieve up to three orders (resp. four orders) of magnitude speed-up over the exact algorithm that computes the BFCC (resp. BTCC) while providing high accuracy with less than 1% relative error.

Scalability test. In this set of experiments, we test the scalability of the proposed algorithms. We adapt the bipartite network model [9] to generate synthetic datasets, where the model is used in the previous studies [84]. We fix the size of the left (resp. right) side vertex set L (resp. R) to four million and generate datasets with different edge set sizes. Fig. 13 (a) (resp. Fig. 13(b)) shows the sampling time of different approximate butterfly (resp. bi-triangle) counting algorithms to get 1% relative error on the synthetic dataset with varying numbers of edges. The running time of an algorithm that failed to achieve at most 1% error within one minute is marked as ∞. The results show that Weighted Pair Sampling (resp. Weighted Triple Sampling) is superior to other algorithms, and all sampling algorithms are insensitive to the size of the graph. We have also done a scalability test by varying the number of vertices. The conclusion is similar, and hence, the results are omitted here. Interested readers are referred to our technical report [2] for more details.

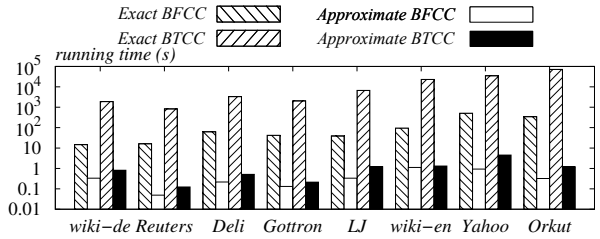
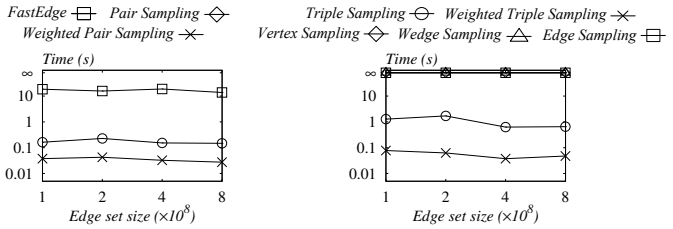


Fig. 12. Time to compute the clustering coefficient: exact algo. v.s. approx. algo. to obtain 1% relative error



(a) Butterfly (b) Bi-triangle

Fig. 13. Scalability test: Varying the number of edges

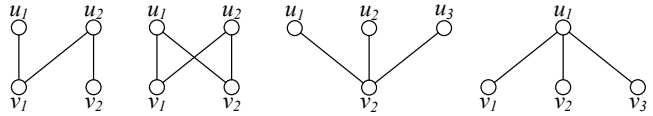
8 CASE STUDIES

In this section, we conduct several case studies to show that the proposed approximate solutions can be applied to compelling real applications.

Graph classification. For the general graph, i.e., unipartite graph, graphlet kernel [51, 59] is a popular method utilized in machine learning to measure the similarity between graphs. The graphlet kernel maps these graphs into a high-dimensional feature space that denotes the frequency of different graphlets. Then, a kernel function measures the similarity between these frequency vectors of graphs within that space. In the literature, the ability of the graphlet kernel to perform graph analysis on general graphs has been widely demonstrated. To investigate the effectiveness of graphlet kernel on bipartite graphs, we collect over 200 real bipartite datasets of different classes from Konect [1], where datasets belong to different categories. Details of the datasets can be found in our repository [2]. For generating the features of bipartite graphs, we extend the graphlet kernel method of general graphs to bipartite graphs, where the graphlet kernel focuses on four-node graphlets. Fig. 14 shows the types of graphlets in bipartite graphs. We use the state-of-the-art exact counting algorithm [76] to generate the exact feature vectors, i.e., graphlet counts. Meanwhile, we also compute approximate feature vectors by using the proposed Weighted Pair Sampling in Sec. 3.2 to estimate the number of butterflies and the approximate 3-path counting algorithm in Sec. 5 to estimate the number of 3-paths. The other two types (Figs. 14(c)-(d)) can be easily estimated by sampling a vertex to quickly derive an unbiased estimation. After obtaining the features, we employ the popular SVM [16] and XGBoost [12] to train a classification model. Tab. 3 shows the results of different methods. We can see that feature generation is time-consuming and the main bottleneck in the exact graphlet feature-based method. The approximate features with the proposed algorithms achieve up to two orders of magnitude speedup over the exact feature generation algorithm. Meanwhile, graph classification models based on approximate features and exact features achieve nearly identical F1 scores. The result shows that the proposed solution can significantly improve the efficiency of graphlet kernel classifiers while retaining the quality of the trained model.

Table 3. Graphlet kernel test results

Graphlet Feature	Classifier	Feature Generation	Training	F1-Score
Exact	SVM	1476s	<1s	0.974
	XGBoost	1476s	<1s	0.987
Approximate	SVM	<10s	<1s	0.973
	XGBoost	<10s	<1s	0.986



(a) (b) (c) (d)

Fig. 14. Features of bipartite graph

Graph clustering. In an existing study [84], the clustering coefficient is used to measure the quality of clusters or communities obtained by different algorithms. In particular, they use the bi-triangle-based clustering coefficient to show that BLP outperforms BRIM, both of which are graph clustering algorithms. However, the algorithm RSWJ-Count used in [84] for calculating the exact clustering coefficient of the clustering results is rather slow, which takes up to three hours to count the number of bi-triangles to obtain the exact value of the clustering coefficients. To put things in context, the clustering algorithms BRIM and BLP themselves obtain the resulting clusters within five minutes on hundreds of millions of scale graphs. In our evaluation, the proposed algorithm can calculate an approximate clustering coefficient in less than half a minute for clustered subgraphs, with no more than 1% relative error. At the same time, using the approximate clustering coefficient leads to the same conclusion as the exact clustering coefficient in Ref. [84]. This use case shows that the approximate solution using the proposed algorithm can significantly reduce the time to measure the quality of graph clustering results with accurate results.

Integrating to NetworkX library. NetworkX [37] is a popular Python library for graph and network algorithms. To help users analyze the properties of bipartite graphs, this library provides functions to compute the bipartite clustering coefficient. But the algorithm used by the network library is relatively inefficient. For example, to examine the effectiveness of proposed algorithms, we use the bipartite graph Reuters as adopted in the previous work [39]. The NetworkX library takes about 50,000 seconds (i.e. 14 hours) to calculate the exact clustering coefficient. In contrast, our Weighted Pair Sampling algorithm, which we implemented and integrated into NetworkX package, spends only 1 second to estimate the clustering coefficient with a relative error of 1%.

9 OTHER RELATED WORK

Motifs [47] are connected graphs consisting of a relatively small number of vertices. Motif counting serves various graph analysis needs, with numerous applications in practice. As a result, motif counting is a well-studied problem in graph data analysis and has received considerable attention [52]. For general, unipartite graphs, the triangle is one of the most fundamental structures, and there have been many studies on triangle counting [8, 24, 28, 32, 62, 64]. For approximate triangle counting, the two most popular methodologies are local sampling and subgraph sampling. Local sampling methods include vertex sampling [3], edge sampling [3], wedge sampling [35], and hybrid sampling [68]. Notably, the Doulion algorithm [67] estimates the number of triangles by a strategy of subgraph sampling, which samples each edge with probability p such that each triangle is

retained with probability p^{-3}. Rasmus et al. [50] propose colorful sampling for subgraph sampling, in which the probability of each triangle being retained increases to p^{-2}.

Bipartite graphs can be regarded as a special case of general graphs. Motif counting and enumeration on bipartite graphs have recently attracted significant research attention. Various studies have explored distinct scenarios of butterfly counting, such as on GPU [81], data streaming [56, 60], uncertain graphs [88], among others. Ref. [76] studies how to maintain the number of butterflies in a batch-dynamic graph based on the previous work [74]. In [55] they also study how to adopt existing subgraph sampling algorithms for approximate triangle counting to approximate butterfly counting. This approach mainly limits the space consumption of the algorithm and is more suitable for streaming scenarios for approximate counting. In the literature, the approximate counting method[67] based on subgraph sampling was found to require sampling a large proportion of the original graph to produce reasonable estimates. The intuitive reason is that subgraph sampling is a more general technique. Compared to subgraph sampling algorithms, local sampling algorithms find a wider range of real-world applications [55]. Hence, this paper focuses on the algorithm based on local sampling. In addition to butterfly and bi-triangle counting, Ref. [82] also studies (p, q)-biclique counting and enumeration. Ref. [87] investigates how to enumerate all bicliques efficiently. Ref. [44] studies how to enumerate maximum biclique. In the context of cohesive subgraph mining, research has been conducted on butterfly-based bi-truss decomposition [75, 77]. Ref. [57] presents a hierarchical structure for modeling dense subgraphs based on the butterfly motif.

10 CONCLUSIONS

In this paper, we propose novel one-sided weighted sampling algorithms for approximate butterfly and bi-triangle counting, which are highly effective and efficient since they capture a large number of motifs in one shot by exploiting the special properties of bipartite graphs. Furthermore, based on a power-law random graph model, we theoretically analyze and show that the proposed solution achieves lower asymptotic complexity compared to the state-of-the-art solution. Besides, we applied the proposed algorithms to estimate clustering coefficients. The high accuracy and efficiency of the proposed algorithm are verified by experiments conducted using several large-scale real bipartite networks. As for future work, we plan to study other types of motif counting on bipartite graphs.

ACKNOWLEDGMENTS

Sibo Wang is supported by the NSFC grant (No. U1936205), the Hong Kong RGC ECS grant (No. 24203419), the RGC GRF grant (No. 14217322), RGC CRF grant (No. C4158-20G) and the Hong Kong ITC ITF grant (No. MRP/071/20X). Junhao Gan is in part supported by ARC Discovery Early Career Researcher Award (DECRA) DE190101118. Yin Yang is supported by the Qatar National Research Fund, a member of the Qatar Foundation (No. NPRP11C-1229-170007). Any opinions, findings, and conclusions, or recommendations expressed in this material are those of the authors and do not reflect the views of the funding agencies.

REFERENCES

[1] 2013. KONECT. http://konect.cc/networks/.
[2] 2023. Code and technical report. https://github.com/CUHK-DBGroup/SIGMOD24-Butterfly-Bi-Triangle-Counting.
[3] Nesreen K. Ahmed, Nick G. Duffield, Jennifer Neville, and Ramana Rao Kompella. 2014. Graph sample and hold: a framework for big-graph analytics. In *KDD*. 1446–1455.
[4] William Aiello, Fan R. K. Chung, and Linyuan Lu. 2000. A random graph model for massive graphs. In *STOC*. 171–180.
[5] Sinan Aksoy, Tamara G. Kolda, and Ali Pinar. 2017. Measuring and modeling bipartite graphs with community structure. *J. Complex Networks* 5, 4 (2017), 581–603.
[6] Michael J Barber. 2007. Modularity and community detection in bipartite networks. *Physical Review E* 76, 6 (2007), 066102.
[7] Rémi Bardenet and Odalric-Ambrym Maillard. 2015. Concentration inequalities for sampling without replacement. *Bernoulli* 21, 3 (2015), 1361–1385.
[8] Luca Becchetti, Paolo Boldi, Carlos Castillo, and Aristides Gionis. 2008. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *KDD*. 16–24.
[9] Etienne Birmelé. 2009. A scale-free graph model based on bipartite graphs. *Discret. Appl. Math.* 157, 10 (2009), 2267–2284.
[10] Stephen P Borgatti and Martin G Everett. 1997. Network analysis of 2-mode data. *Social networks* 19, 3 (1997), 243–269.
[11] Sudarshan S. Chawathe and Hector Garcia-Molina. 1997. Meaningful Change Detection in Structured Data. In *SIGMOD*. 26–37.
[12] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *SIGKDD*. 785–794.
[13] Xingguang Chen and Sibo Wang. 2021. Efficient Approximate Algorithms for Empirical Entropy and Mutual Information. In *SIGMOD*. 274–286.
[14] Xingguang Chen, Fangyuan Zhang, and Sibo Wang. 2022. Efficient Approximate Algorithms for Empirical Variance with Hashed Block Sampling. In *SIGKDD*. 157–167.
[15] Norishige Chiba and Takao Nishizeki. 1985. Arboricity and subgraph listing algorithms. *SIAM Journal on computing* 14, 1 (1985), 210–223.
[16] Corinna Cortes and Vladimir Vapnik. 1995. Support-Vector Networks. *Mach. Learn.* 20, 3 (1995), 273–297.
[17] Hongbo Deng, Michael R. Lyu, and Irwin King. 2009. A generalized Co-HITS algorithm and its application to bipartite graphs. In *SIGKDD*. 239–248.
[18] Inderjit S. Dhillon. 2001. Co-clustering documents and words using bipartite spectral graph partitioning. In *SIGKDD*. 269–274.
[19] Yixiang Fang, Xin Huang, Lu Qin, Ying Zhang, Wenjie Zhang, Reynold Cheng, and Xuemin Lin. 2020. A survey of community search over big graphs. *VLDB J.* 29, 1 (2020), 353–392.
[20] Yixiang Fang, Kaiqiang Yu, Reynold Cheng, Laks V. S. Lakshmanan, and Xuemin Lin. 2019. Efficient Algorithms for Densest Subgraph Discovery. *Proc. VLDB Endow.* 12, 11 (2019), 1719–1732.
[21] Xiaoli Zhang Fern and Carla E. Brodley. 2004. Solving cluster ensemble problems by bipartite graph partitioning. In *ICML*.
[22] Qintian Guo, Sibo Wang, Zhewei Wei, and Ming Chen. 2020. Influence Maximization Revisited: Efficient Reverse Reachable Set Generation with Bound Tightened. In *SIGMOD*. 2167–2181.
[23] Qintian Guo, Sibo Wang, Zhewei Wei, Wenqing Lin, and Jing Tang. 2022. Influence Maximization Revisited: Efficient Sampling with Bound Tightened. *ACM Trans. Database Syst.* 47, 3 (2022), 12:1–12:45.
[24] Mohammad Al Hasan and Vachik S. Dave. 2018. Triangle counting in large networks: a review. *WIREs Data Mining Knowl. Discov.* 8, 2 (2018).
[25] Paul W Holland and Samuel Leinhardt. 1976. Local structure in social networks. *Sociological methodology* 7 (1976), 1–45.
[26] Bryan Hooi, Hyun Ah Song, Alex Beutel, Neil Shah, Kijung Shin, and Christos Faloutsos. 2016. FRAUDAR: Bounding Graph Fraud in the Face of Camouflage. In *SIGKDD*. 895–904.
[27] Guanhao Hou, Qintian Guo, Fangyuan Zhang, Sibo Wang, and Zhewei Wei. 2023. Personalized PageRank on Evolving Graphs with an Incremental Index-Update Scheme. *Proc. ACM Manag. Data* 1, 1 (2023), 25:1–25:26.
[28] Xiaocheng Hu, Yufei Tao, and Chin-Wan Chung. 2013. Massive graph triangulation. In *SIGMOD*. 325–336.
[29] Chu-Yi Huang, Yen-Shen Chen, Youn-Long Lin, and Yu-Chin Hsu. 1990. Data Path Allocation Based on Bipartite Weighted Matching. In *DAC*. IEEE Computer Society Press, 499–504.
[30] Xin Huang, Hong Cheng, Lu Qin, Wentao Tian, and Jeffrey Xu Yu. 2014. Querying k-truss community in large and dynamic graphs. In *SIGMOD*. 1311–1322.
[31] Xin Huang, Wei Lu, and Laks V. S. Lakshmanan. 2016. Truss Decomposition of Probabilistic Graphs: Semantics and Algorithms. In *SIGMOD*. 77–90.
[32] Alon Itai. 1977. Finding a Minimum Circuit in a Graph. In *STOC*. 1–10.

[33] Mark Jerrum, Leslie G. Valiant, and Vijay V. Vazirani. 1986. Random Generation of Combinatorial Structures from a Uniform Distribution. *Theor. Comput. Sci.* 43 (1986), 169–188.
[34] Yuli Jiang, Yu Rong, Hong Cheng, Xin Huang, Kangfei Zhao, and Junzhou Huang. 2022. Query Driven-Graph Neural Networks for Community Search: From Non-Attributed, Attributed, to Interactive Attributed. *Proc. VLDB Endow.* 15, 6 (2022), 1243–1255.
[35] Tamara G. Kolda, Ali Pinar, and C. Seshadhri. 2013. Triadic Measures on Graphs: The Power of Wedge Sampling. In *SDM*. 10–18.
[36] Jérôme Kunegis. 2013. KONECT: the Koblenz network collection. In *WWW*. 1343–1350.
[37] Los Alamos National Laboratory. 2023. Networkx. https://networkx.org/.
[38] Matthieu Latapy, Clémence Magnien, and Nathalie Del Vecchio. 2008. Basic notions for the analysis of large two-mode networks. *Social networks* 30, 1 (2008), 31–48.
[39] David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. 2004. RCV1: A New Benchmark Collection for Text Categorization Research. *J. Mach. Learn. Res.* 5 (2004), 361–397.
[40] Feifei Li, Bin Wu, Ke Yi, and Zhuoyue Zhao. 2016. Wander Join: Online Aggregation via Random Walks. In *SIGMOD*. 615–629.
[41] Pedro G Lind, Marta C González, and Hans J Herrmann. 2005. Cycles and clustering in bipartite networks. *Physical review E* 72, 5 (2005), 056127.
[42] Boge Liu, Long Yuan, Xuemin Lin, Lu Qin, Wenjie Zhang, and Jingren Zhou. 2019. Efficient (α,β)-core Computation: an Index-based Approach. In *WWW*. 1130–1141.
[43] Xin Liu and Tsuyoshi Murata. 2009. Community Detection in Large-Scale Bipartite Networks. In *Web Intelligence*. 50–57.
[44] Bingqing Lyu, Lu Qin, Xuemin Lin, Ying Zhang, Zhengping Qian, and Jingren Zhou. 2020. Maximum Biclique Search at Billion Scale. *Proc. VLDB Endow.* 13, 9 (2020), 1359–1372.
[45] Mohammad Mahdian and Qiqi Yan. 2011. Online bipartite matching with random arrivals: an approach based on strongly factor-revealing LPs. In *STOC*. 597–606.
[46] Charles Masson, Jee E. Rim, and Homin K. Lee. 2019. DDSketch: A Fast and Fully-Mergeable Quantile Sketch with Relative-Error Guarantees. *Proc. VLDB Endow.* 12, 12 (2019), 2195–2205.
[47] Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. 2002. Network motifs: simple building blocks of complex networks. *Science* 298, 5594 (2002), 824–827.
[48] Tore Opsahl. 2013. Triadic closure in two-mode networks: Redefining the global and local clustering coefficients. *Soc. Networks* (2013), 159–167.
[49] Michael D Ornstein. 1982. Interlocking directorates in Canada: evidence from replacement patterns. *Social Networks* 4, 1 (1982), 3–25.
[50] Rasmus Pagh and Charalampos E. Tsourakakis. 2012. Colorful triangle counting and a MapReduce implementation. *Inf. Process. Lett.* 112, 7 (2012), 277–281.
[51] Nataša Pržulj. 2007. Biological network comparison using graphlet degree distribution. *Bioinformatics* 23, 2 (2007), e177–e183.
[52] Pedro Ribeiro, Pedro Paredes, Miguel EP Silva, David Aparicio, and Fernando Silva. 2021. A survey on subgraph counting: concepts, algorithms, and applications to network motifs and graphlets. *ACM Computing Surveys (CSUR)* 54, 2 (2021), 1–36.
[53] Garry Robins and Malcolm Alexander. 2004. Small Worlds Among Interlocking Directors: Network Structure and Distance in Bipartite Graphs. *Comput. Math. Organ. Theory* 10, 1 (2004), 69–94.
[54] Boyu Ruan, Junhao Gan, Hao Wu, and Anthony Wirth. 2021. Dynamic Structural Clustering on Graphs. In *SIGMOD*. 1491–1503.
[55] Seyed-Vahid Sanei-Mehri, Ahmet Erdem Sariyüce, and Srikanta Tirthapura. 2018. Butterfly Counting in Bipartite Networks. In *SIGKDD*. 2150–2159.
[56] Seyed-Vahid Sanei-Mehri, Yu Zhang, Ahmet Erdem Sariyüce, and Srikanta Tirthapura. 2019. FLEET: Butterfly Estimation from a Bipartite Graph Stream. In *CIKM*. 1201–1210.
[57] Ahmet Erdem Sariyüce and Ali Pinar. 2018. Peeling Bipartite Networks for Dense Subgraph Discovery. In *WSDM*. 504–512.
[58] Thomas Schank and Dorothea Wagner. 2005. Approximating Clustering Coefficient and Transitivity. *J. Graph Algorithms Appl.* 9, 2 (2005), 265–275.
[59] Nino Shervashidze, S. V. N. Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten M. Borgwardt. 2009. Efficient graphlet kernels for large graph comparison. In *AISTATS (JMLR Proceedings, Vol. 5)*. 488–495.
[60] Aida Sheshbolouki and M. Tamer Özsu. 2022. sGrapp: Butterfly Approximation in Streaming Graphs. *ACM Trans. Knowl. Discov. Data* 16, 4 (2022), 76:1–76:43.
[61] Jessica Shi and Julian Shun. 2020. Parallel Algorithms for Butterfly Computations. In *APOCS*. SIAM, 16–30.

[62] Julian Shun and Kanat Tangwongsan. 2015. Multicore triangle computations without tuning. In *ICDE*. 149–160.
[63] Jimeng Sun, Huiming Qu, Deepayan Chakrabarti, and Christos Faloutsos. 2005. Neighborhood Formation and Anomaly Detection in Bipartite Graphs. In *ICDM*. 418–425.
[64] Siddharth Suri and Sergei Vassilvitskii. 2011. Counting triangles and the curse of the last reducer. In *WWW*. 607–614.
[65] Amos Tanay, Roded Sharan, and Ron Shamir. 2002. Discovering statistically significant biclusters in gene expression data. *Bioinformatics* 18, suppl_1 (2002), S136–S144.
[66] Youze Tang, Yanchen Shi, and Xiaokui Xiao. 2015. Influence Maximization in Near-Linear Time: A Martingale Approach. In *SIGMOD*. 1539–1554.
[67] Charalampos E. Tsourakakis, U Kang, Gary L. Miller, and Christos Faloutsos. 2009. DOULION: counting triangles in massive graphs with a coin. In *SIGKDD*. 837–846.
[68] Duru Türkoglu and Ata Turk. 2017. Edge-Based Wedge Sampling to Estimate Triangle Counts in Very Large Graphs. In *ICDM*. 455–464.
[69] Johan Ugander, Lars Backstrom, and Jon M. Kleinberg. 2013. Subgraph frequencies: mapping the empirical and extremal geography of large graph collections. In *WWW*. 1307–1318.
[70] Demival Vasques Filho and Dion RJ O'Neale. 2018. Degree distributions of bipartite networks and their projections. *Physical Review E* 98, 2 (2018), 022307.
[71] Alastair J. Walker. 1977. An Efficient Method for Generating Discrete Random Variables with General Distributions. *ACM Trans. Math. Softw.* 3, 3 (1977), 253–256.
[72] Jia Wang, Ada Wai-Chee Fu, and James Cheng. 2014. Rectangle Counting in Large Bipartite Graphs. In *IEEE International Congress on Big Data*. 17–24.
[73] Kai Wang, Yiheng Hu, Xuemin Lin, Wenjie Zhang, Lu Qin, and Ying Zhang. 2021. A Cohesive Structure Based Bipartite Graph Analytics System. In *CIKM*. 4799–4803.
[74] Kai Wang, Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang. 2019. Vertex Priority Based Butterfly Counting for Large-scale Bipartite Networks. *Proc. VLDB Endow.* (2019), 1139–1152.
[75] Kai Wang, Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang. 2020. Efficient Bitruss Decomposition for Large-scale Bipartite Graphs. In *ICDE*. 661–672.
[76] Kai Wang, Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang. 2022. Accelerated butterfly counting with vertex priority on bipartite graphs. *VLDB J.* (2022).
[77] Kai Wang, Xuemin Lin, Lu Qin, Wenjie Zhang, and Ying Zhang. 2022. Towards efficient solutions of bitruss decomposition for large-scale bipartite graphs. *VLDB J.* 31, 2 (2022), 203–226.
[78] Sibo Wang, Youze Tang, Xiaokui Xiao, Yin Yang, and Zengxiang Li. 2016. HubPPR: Effective Indexing for Approximate Personalized PageRank. *Proc. VLDB Endow.* 10, 3 (2016), 205–216.
[79] Sibo Wang, Renchi Yang, Xiaokui Xiao, Zhewei Wei, and Yin Yang. 2017. FORA: Simple and Effective Approximate Single-Source Personalized PageRank. In *SIGKDD*. 505–514.
[80] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. 2019. Neural Graph Collaborative Filtering. In *SIGIR*. 165–174.
[81] Qingyu Xu, Feng Zhang, Zhiming Yao, Lv Lu, Xiaoyong Du, Dong Deng, and Bingsheng He. 2022. Efficient Load-Balanced Butterfly Counting on GPU. *Proc. VLDB Endow.* 15, 11 (2022), 2450–2462.
[82] Jianye Yang, Yun Peng, and Wenjie Zhang. 2021. (p, q)-biclique Counting and Enumeration for Large Sparse Bipartite Graphs. *Proc. VLDB Endow.* 15, 2 (2021), 141–153.
[83] Yixing Yang, Yixiang Fang, Xuemin Lin, and Wenjie Zhang. 2020. Effective and Efficient Truss Computation over Large Heterogeneous Information Networks. In *ICDE*. 901–912.
[84] Yixing Yang, Yixiang Fang, Maria E. Orlowska, Wenjie Zhang, and Xuemin Lin. 2021. Efficient Bi-triangle Counting for Large Bipartite Networks. *Proc. VLDB Endow.* (2021), 984–996.
[85] Fangyuan Zhang, Mengxu Jiang, and Sibo Wang. 2023. Efficient Dynamic Weighted Set Sampling and Its Extension. *Proc. VLDB Endow.* 17, 1 (2023), 15–27.
[86] Fangyuan Zhang and Sibo Wang. 2022. Effective Indexing for Dynamic Structural Graph Clustering. *Proc. VLDB Endow.* 15, 11 (2022), 2908–2920.
[87] Yun Zhang, Charles A Phillips, Gary L Rogers, Erich J Baker, Elissa J Chesler, and Michael A Langston. 2014. On finding bicliques in bipartite graphs: a novel algorithm and its application to the integration of diverse biological data types. *BMC bioinformatics* 15, 1 (2014), 1–18.
[88] Alexander Zhou, Yue Wang, and Lei Chen. 2021. Butterfly Counting on Uncertain Bipartite Networks. *Proc. VLDB Endow.* 15, 2 (2021), 211–223.
[89] Tao Zhou, Jie Ren, Matúš Medo, and Yi-Cheng Zhang. 2007. Bipartite network projection and personal recommendation. *Physical review E* 76, 4 (2007), 046115.

Received April 2023; revised July 2023; accepted August 2023