ProbGraph: High-Performance and High-Accuracy Graph Mining with Probabilistic Set Representations

Maciej Besta\textsuperscript{1,}\textsuperscript{†}, Cesare Miglioli\textsuperscript{2}, Paolo Sylos Labini\textsuperscript{3}, Jakub Tětek\textsuperscript{4}, Patrick Iffi\textsuperscript{1}, Raghavendra Kanakagiri\textsuperscript{5}, Saleh Ashkboos\textsuperscript{1}, Kacper Janda\textsuperscript{6}, Michał Podstawski\textsuperscript{7,8}, Grzegorz Kwaśniewski\textsuperscript{1}, Niels Gleiming\textsuperscript{1}, Flavio Vella\textsuperscript{9}, Onur Mutlu\textsuperscript{10}, Torsten Hoefler\textsuperscript{11}

\textsuperscript{1}ETH Zurich, Zürich, Switzerland \textsuperscript{2}Research Center for Statistics, University of Geneva, Geneva, Switzerland \textsuperscript{3}Free University of Bozen-Bolzano, Bolzano, Italy \textsuperscript{4}BARC, University of Copenhagen, Copenhagen, Denmark \textsuperscript{5}UIUC, Champaign, USA \textsuperscript{6}AGH-UST, Kraków, Poland \textsuperscript{7}Warsaw University of Technology, Warsaw, Poland \textsuperscript{8}TCL Research Europe, Warsaw, Poland \textsuperscript{9}University of Trento, Trento, Italy

Corresponding authors: maciej.best@inf.ethz.ch, htor@inf.ethz.ch

Abstract—Important graph mining problems such as Clustering are computationally demanding. To significantly accelerate these problems, we propose ProbGraph: a graph representation that enables simple and fast approximate parallel graph mining with strong theoretical guarantees on work, depth, and result accuracy. The key idea is to represent sets of vertices using probabilistic set representations such as Bloom filters. These representations are much faster to process than the original vertex sets thanks to vectorizability and small size. We use these representations as building blocks in important parallel graph mining algorithms such as Clique Counting or Clustering. When enhanced with ProbGraph, these algorithms significantly outperform tuned parallel exact baselines (up to nearly 50× on 32 cores) while ensuring accuracy of more than 90% for many input graph datasets. Our novel bounds and algorithms based on probabilistic set representations with desirable statistical properties are of separate interest for the data analytics community.

Index Terms—Approximate Graph Mining, Approximate Graph Pattern Matching, Approximate Triangle Counting, Approximate Community Detection, Approximate Graph Clustering, Bloom Filters, MinHash, K Minimum Values, High-Performance Graph Computations, Graph Sketching

I. INTRODUCTION

Graph mining is an important part of the graph processing landscape, with problems related to discovering patterns in graphs, for example Clustering, Clique Counting, or Link Prediction [1]–[4]. Accelerating graph mining is notoriously difficult because these problems are hard to parallelize due to properties such as high irregularity or little locality [5]–[10]. Simultaneously, graph mining underlies many important computational problems in social network analysis, machine learning, computational sciences, and others [11]–[14].

In approximate computing, a certain (ideally small) amount of accuracy is sacrificed, in exchange for speedups or reduced energy consumption [15]–[17]. It relaxes the need for full precision at the level of arithmetic blocks, processing units, pertinent error and quality measures, algorithms, programming models, and many others. Traditionally, graph algorithms with provable approximation ratios were developed to alleviate the hardness of various NP-Complete graph problems, such as minimum graph coloring [18], [19]. Still, these works are usually complex, specific to a particular graph problem, and often need additional heuristics to be easily used in practice.

Moreover, there are many heuristics for approximating graph properties such as betweenness centrality [20]–[25], minimum spanning tree weight [26], maximum matching [27], reachability [28], graph diameter [29], [30], and others [30]–[34]. Unfortunately, these schemes are all specific to a particular graph problem or algorithm.

To alleviate the above-mentioned issues with accelerating graph mining, we propose ProbGraph (PG), a probabilistic graph representation for simple, versatile, fast, and tunable approximate graph mining. We observe that the input graph and many auxiliary data structures are effectively a collection of sets of vertices and edges [1], [4]. Here, our key idea is to encode such sets with carefully selected probabilistic set representations (sometimes called set sketches) such as Bloom filters [35]. This results in a “probabilistic” graph representation that – as we will show – can accelerate different graph algorithms at a (tunable) accuracy-storage-performance tradeoff. Importantly, sets and set operations are common in graph problems, making PG applicable to many algorithms.

We first show that many time-consuming operations in different graph algorithms can be expressed with set intersection cardinality $|X \cap Y|$. For example, deriving common parts of vertex neighborhoods takes more than 90% of the time in common Triangle Counting algorithms [1], [4], [36], and it can be expressed as a sum of $|X \cap Y|$ (over different $X$ and $Y$). We identify more such graph algorithms.

Second, we carefully select three probabilistic set representations: Bloom filters (BF) [35] and two types of MinHash (MH) [37]. We use these set representations to design estimators $|X \cap Y|$ that approximate $|X \cap Y|$. Our central motivation is that these estimators are much faster to obtain than the exact $|X \cap Y|$ thanks to performance-friendly properties. We conduct a work-depth analysis, showing formally that ProbGraph-enhanced graph algorithms have abundant parallelism. For example, $X$ and $Y$, when represented with Bloom filters, are bit vectors. Thus, $|X \cap Y|$ amounts to computing a bitwise AND, followed by a reduction. Such an...
operation significantly benefits from vectorization. Moreover, thanks to its fixed-size set representations, ProbGraph exhibits excellent load balancing properties: all set intersections are conducted over the same size bit vectors, annihilating issues related to intersecting neighborhoods of different sizes. This is particularly attractive as modern graph datasets have very often high skews in degree distributions [7].

Importantly, we ensure that our estimators have strong theoretical guarantees on their accuracy (i.e., quality). For this, we develop or adapt bounds on the quality of estimators $|X \cap Y|$. Here, we prove how far they deviate from the true size of the set intersection, and we provide upper bounds for their mean squared error (MSE). We also produce quality bounds that are better than past works. To derive these quality bounds, we use concentration inequalities and other statistical concepts such as sub-Gaussian random variables [38]. For example, the probability that our estimator based on MH deviates from the true value by more than a given distance $d$, decreases exponentially with $t$.

Moreover, we show that, for some representations, we offer Maximum Likelihood Estimators (MLE) [39]. Thus, these estimators are asymptotically unbiased (the expected estimator value converges to the true parameter value) and efficient (asymptotically, no other estimator has lower variance). Due to the prevalence of BF and MH in high-performance data mining, our novel results on the theory of probabilistic set representations are of interest beyond graph analytics.

Our fast parallel implementation of PG enables important graph mining problems (TC, clustering [40]-[42], 4-clique counting [43], and vertex similarity [44], [45]) to achieve very large performance advantages over tuned baselines, even up to $50 \times$ lower runtimes when using 32 cores. This is caused by the fact that both work and depth of ProbGraph-enhanced graph algorithms are lower than those of the exact tuned baselines. Simultaneously, using small fixed-size probabilistic representations makes it much easier to load balance expensive set operations. Simultaneously, ProbGraph achieves high accuracy of more than 90% for many inputs.

We also provide strong theoretical guarantees on work and depth of all the considered graph mining algorithms. Finally, we use our $|X \cap Y|$ estimators to derive novel estimators $TC$ on the triangle count $TC$ in an arbitrary graph, achieving strong statistical properties such as MLE and exponential concentration quality bounds. Our $TC$ estimator based on MH has better theoretical properties (e.g., is MLE) than past theoretical results [46]-[54].

Overall, PG enables trading a small amount of accuracy and storage for more performance. These tradeoffs are tunable by the user, who can select which aspect is most important.

II. FUNDAMENTAL CONCEPTS

We first present the used concepts and symbols (see Table I).

A. Graph Model and Representation

A graph $G$ is modeled as a tuple $(V, E)$ with a set of vertices $(V, |V| = n)$ and edges $(E \subseteq V \times V, |E| = m)$. We model vertices with their integer IDs $(V = \{1, \ldots, n\})$, $N_v$ and $d_v$ denote the neighbors and the degree of a given vertex $v$ $(N_v \subset V)$; $d$ is $G$’s maximum degree. We store the input (i.e., not sketched) graph $G$ using the standard Compressed Sparse Row (CSR) format, in which all neighborhoods $N_v$ form a contiguous array (2m words if $G$ is undirected). Next, there is an array with $n$ pointers to each representation of $N_v$. Each $N_v$ is stored as a contiguous sorted array of vertex IDs.

B. Work-Depth Analysis of Parallel Algorithms

We use work-depth (WD) analysis for bounding run-times of parallel algorithms, to analyze PG set intersections (§ IV-F, Table IV), construction costs (§ VI-A, Table V), and graph algorithms (§ VI-B, Table VI). The work of an algorithm is the total number of operations and the depth is the longest sequential chain of execution in the algorithm (assuming infinite number of parallel threads executing the algorithm) [55], [56].

C. Set Algebra

When using arbitrary sets, we use symbols $X = \{x_1, \ldots, x_l\}$ and $Y = \{y_1, \ldots, y_l\}$. We use intersection $(X \cap Y)$, union $(X \cup Y)$, cardinality ($|X|$), and membership ($x \in X$).

D. Probabilistic Set Representations

We consider Bloom filters (BF) and two variants of MinHash (MH). We pick different representations to better understand which ones are best suited for accelerating graph mining problems with high accuracy. All the proofs of theorems in the following sections are in the appendix. Figure 1 illustrates the representations considered.

**Bloom filters** The Bloom filter (BF) [35] is a space-efficient set representation that answers membership queries fast but with some probability of false positives. Formally, a Bloom filter $B_X$ representing a set $X$ consists of an $l$-element bit vector $B_X$ (initialized to zeros) and $b$ hash functions $h_1, \ldots, h_b$ (usually assumed to be independent) that map elements of $X$ to integers in $[l]$ ($[l] = \{1, \ldots, l\}$). The size of $B_X$ is also denoted with $B_X$ while the number of ones in $B_X$ with $B_{X,1}$. Now, when constructing $B_X$, for each element $x \in X$, one computes the corresponding hashes $h_1(x), \ldots, h_b(x)$.

Then, the bits $B_X[h_1(x)]$, ..., $B_X[h_b(x)]$ are set to one. Second, verifying if $x \in X$ is similar. First, all hash functions are evaluated for $x$. If all bits at the corresponding positions are set, i.e., $\forall i \in \{1, \ldots, b\}: B_X[h_i(x)] = 1$, then $x$ is considered to be in $X$. It is possible that some of these bits are set to one while adding other elements due to hash collisions, and $x$ might be falsely reported as being an element of $X$. Minimizing the number of such false positives was addressed in many research works [57].

**MinHash** (k-Hash variant) MinHash (MH) [58] “sketches” a set $X$ by hashing its elements to integers and keeping $k$ elements with smallest hashes. An MH representation $M_X$ of $X$ consists of a set $M_X$ with elements from $X$ ($\forall x \in M_X \ x \in X$)
and k hash functions \( h_i, i \in \{1, \ldots, k\} \). To construct \( \mathcal{M}_X \), one computes all hash values \( h_i(x) \) for each \( x \in X \). Then, for each hash function \( h_i \) separately, one selects an element \( x_{i, \min} \in X \) that has the smallest hash \( h_i(x) \). These elements form the final set \( X_{\max} = \{ x_{i, \min}, \ldots, x_{k, \min} \} \). Note that \( X_{\max} \) may be a multi set: if \( x_{i, \min} = x_{j, \min} \) for \( i \neq j \), then \( X_{\max} \) contains \( x_{i, \min} \) twice.

MinHash (1-Hash variant) The \( k \)-hash variant may be computationally expensive as it computes \( k \) different hash functions for all elements of \( X \). To alleviate this, one can use a variant called 1-Hash \( (M^1_X) \) with only one hash function \( h \).

After computing hashes \( h(x) \) for all \( x \in X \), \( M^1_X \) will contain \( k \) elements from \( X \) that resulted in \( k \) smallest hash values. Unlike \( k \)-hash, \( M^1_X \) by definition never contains duplicates.

### E. Estimators

A central concept in PG is an estimator: a rule for calculating an estimate of a given quantity based on observed data. We develop estimators of set sizes \( |X| \) and set intersection cardinalities \( |X \cap Y| \), and we use them to approximate \( |N_v \cap N_u| \) for any two vertices \( u, v \) in considered graph algorithms.

We describe in more detail a specific BF estimator by Swamidass et al. [59]. We later generalize it and also prove novel bounds on its quality. Given a set \( X \) represented by a Bloom filter \( B_X \), one can estimate \( |X| \) as

![Fig. 1: Overview of selected PG set representations, how they are used to accelerate intersections of vertex neighborhoods, and in alleviating load imbalance.](image)

In panel 1, we show a part of an example input graph. In panel 2, we illustrate a traditional exact way to compute the count of shared neighbors \( |N_u \cap N_v| \) of any vertices \( u \) and \( v \). There are two variants of this operation: “merge” (the upper sub-panel), where one simply merges two sorted sets (it is more beneficial when sets have similar sizes), and “galloping” (the lower sub-panel), where – for each element from a smaller set, one uses binary search to check if this element is in the larger set (it works better when one set is much larger than the other one). In panel 3, we show how to compute \( |N_u \cap N_v| \) with BF. Here, \( N_u \) and \( N_v \) are first converted into bit vectors (cf. wide vertical arrows). Then, the resulting bit vectors are intersected with a very fast AND operation (cf. wide horizontal arrows). In panel 4, we show how to compute \( |N_u \cap N_v| \) with MH. Here, \( N_u \) and \( N_v \) are first appropriately hashed into much smaller subsets of \( N_u \) and \( N_v \) (cf. wide vertical arrows). The resulting sets can be intersected fast because they are much smaller than the original \( N_u \) and \( N_v \) (indicated with wide horizontal arrows). In panel 5, we show load balancing benefits from using PB (i.e., it is easy to load balance intersections of same-sized PG neighborhoods).

<table>
<thead>
<tr>
<th>Graph code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( G = (V, E) )</td>
<td>A graph; ( V, E ) are sets of vertices and edges, respectively.</td>
</tr>
<tr>
<td>( n, m )</td>
<td>Numbers of vertices and edges in ( G ); (</td>
</tr>
<tr>
<td>( d_v, N_v )</td>
<td>The degree and neighbors of ( v \in V ).</td>
</tr>
<tr>
<td>( d, \bar{d} )</td>
<td>The maximum and the average degree in ( G ) (( \bar{d} = m/n )).</td>
</tr>
<tr>
<td>( TC )</td>
<td>Count of triangles in a given graph.</td>
</tr>
<tr>
<td>( W )</td>
<td>Size of a memory word [bits].</td>
</tr>
<tr>
<td>( s )</td>
<td>Storage budget, i.e., space dedicated to PG structures.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>BF</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( B_X, B_X \cdot B_X[1] )</td>
<td>A Bloom filter; the associated bit vector; the ( i )-th bit in ( B_X ).</td>
</tr>
<tr>
<td>( b )</td>
<td>The size of ( B_X ) (#bits); number of ones and zeros in ( B_X ).</td>
</tr>
<tr>
<td>( h_i )</td>
<td>An ( i )-th associated hash function, ( i \in {1, \ldots, b} ).</td>
</tr>
<tr>
<td>( p_f )</td>
<td>The probability of a false positive of a given Bloom filter.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MinHash</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( M^1_X, M_X )</td>
<td>( k )-Hash variant; the approximating set based on ( X ).</td>
</tr>
<tr>
<td>( k )</td>
<td>The number of elements stored in a given MinHash.</td>
</tr>
<tr>
<td>( h_i )</td>
<td>An ( i )-th associated hash function, ( i \in {1, \ldots, k} ).</td>
</tr>
<tr>
<td>( M^1_X, M_X )</td>
<td>1-Hash variant; the approximating set based on ( X ).</td>
</tr>
</tbody>
</table>

TABLE I: Important used symbols. “\( X \)” denotes the input set approximated by a respective representation (omitted if it is clear from context).
\[ \overline{X}_S = -\frac{B_X}{b} \log \left( 1 - \frac{B_{X,1}}{B_X} \right). \]  

To derive this estimator, consider inserting a single element into a given BF \( B_X \). The probability that some single bit equals 0, in \( B_X \) with one hash function (that maps elements uniformly over the bit array), is \( 1 - 1/B_X \). When applying \( b \) hash functions (which follow the usual assumption of being independent), this probability is \((1 - 1/B_X)^b\). After inserting \(|X|\) elements into \( B_X \), the probability that this one bit is 0 is \((1 - 1/B_X)^{|X|} \approx \exp(-b|X|/B_X)\) (from the established identity for \( e^{-1} \approx (1 - 1/x)^x \)). Thus, the probability that this bit is set to 1, is \(1 - (1 - 1/B_X)^{|X|} \approx 1 - \exp(-b|X|/B_X)\). Then, observe that the number of bits set to 1 in \( B_X \), can be estimated as \( B_{X,1} \approx B_X \cdot (1 - \exp(-b|X|/B_X)) \) given the binomial density approximation used in Swamidass et al. [59]. When resolving this equation for \(|X|\), we obtain Eq. (1).

**F. Properties of Estimators**

Figure 2 provides an intuitive description of the desirable statistical properties of PG. These properties enable highly accurate empirical results (Section VIII) and attractive theoretical results for triangle count (Section VII).

All PG estimators are **asymptotically unbiased**. In such an estimator \( \hat{\theta} \), the difference between \( \theta \)'s expected value and the true value of the parameter being estimated \( \theta \) converges to 0 for a fixed input and the size of the sketch that we use going to infinity (i.e., the bias of \( \hat{\theta} \) goes to 0, or, on average, \( \hat{\theta} \) hits \( \theta \) when the sample size approaches the limit). Unbiased estimators are usually more desirable than biased ones: intuitively, they ensure zero average error (when estimating \( \theta \)) after a given amount of trials. Next, each PG estimator of \( \theta \) is also **consistent**, i.e., the sampling distribution of \( \hat{\theta} \) becomes increasingly more concentrated at \( \theta \) with the increasing number of samples. Hence, if there are enough observations (in our case when the sketches are large enough), one can find \( \theta \) with arbitrary precision. Asymptotic unbiasedness alone does not imply consistency; it requires also a vanishing variance (i.e., that the estimator variance converges to 0 with the increasing sample size).

We also verify if PG estimators are maximum likelihood.

This class of estimators provides several powerful and useful properties, and is among the most important tools for estimation and inference in statistics [60]–[62]. Specifically, a maximum likelihood estimator (MLE) \( \hat{\theta}_{MLE} \) is an estimator that maximizes the **likelihood function** \( L \), i.e., \( \hat{\theta}_{MLE} = \arg\max_{\theta \in \Theta} L(x; \theta) \) (cf., Chapter 7 in [39]) where \( \Theta \) is the parameter space. Here, the likelihood function \( L \) is defined as the probability of observing a given sample \( x = (x_1, ..., x_n) \) as a function of \( \theta \), i.e., \( L = P(X| \theta) = x_1, ..., x_n; \theta \), where \( X_1, ..., X_n \) represent a random sample from a given population. Thus, \( \hat{\theta}_{MLE} \) is the value of the parameter \( \theta \) for which the observed sample is the most likely. This intuitive choice of an estimator leads under mild conditions to strong optimality properties such as consistency (discussed above), invariance, and **asymptotic efficiency**. An estimator \( \hat{\theta} \) is invariant if, whenever \( \theta \) is the MLE of \( \theta \), then for any function \( \tau(\cdot) \), \( \tau(\hat{\theta}) \) is the MLE of \( \tau(\theta) \). This is useful if complicated functions of the parameter \( \theta \) are of interest since \( \tau(\hat{\theta}_{MLE}) \) inherits automatically all the properties of \( \hat{\theta}_{MLE} \). Finally, the asymptotic efficiency certifies that \( \hat{\theta}_{MLE} \) attains, under mild regularity conditions, the Cramer-Rao bound [39]. To understand the importance of this result, we need to introduce the Mean Squared Error (MSE) of an estimator. MSE measures the average squared difference between the estimator and the parameter, i.e., \( MSE(\hat{\theta}, \theta) = E_\theta[(\hat{\theta} - \theta)^2] \). Thus, the asymptotic efficiency implies that there exists no consistent estimator of \( \theta \) that achieves a lower MSE than \( \hat{\theta}_{MLE} \). Next, given the well known decomposition of the MSE into bias squared plus variance (i.e., \( MSE(\hat{\theta}) = Bias(\hat{\theta}, \theta)^2 + Var_\theta(\hat{\theta}) \)), we can also conclude that among all the consistent estimators of \( \theta \), no one has a strictly smaller variance than \( \hat{\theta}_{MLE} \). Another interesting implication of this result is the **asymptotic normality** of MLE. This property allows to build valid confidence intervals for the parameter of interest based on the normal distribution (asymptotically in sketch size, for a fixed input).

**G. Concentration Bounds**

We use the notion of a **concentration inequality**. Overall, such an inequality bounds the deviation of a given random variable \( X \) from some value (usually the expectation \( E[X] \)). In this work, we mainly use the Chebyshev [38], Hoeffding [63] and Serfling inequalities [64].

**III. Sets & \(|X \cap Y|\) in Graph Algorithms**

We first identify algorithms that use \(|X \cap Y|\). A graph itself can be modeled as a collection of sets: each vertex neighborhood \( N_v \) is essentially a set. In PG, we use this observation to approximate the graph structure and operations by using probabilistic set representations in place of \( N_v \) and \(|N_v \cap N_u|\), for any vertices \( u \) and \( v \). In the following listings, the “\( X \)” and “\( Y \)” general symbols are replaced with specific sets. Operations approximated by PG are marked with the blue color. “[in par]” means that a loop is parallelized. We ensure that the parallelization does not involve conflicting memory accesses. For clarity, we focus on formulations and we discuss parallelization details in Section VI.

**In Triangle Counting (TC) [65], [66] (Listing 1), one counts the total number of triangles \( tc \) in an undirected graph. An example application of TC is computing clustering coefficients [67]. For each vertex \( u \), one computes the cardinalities of the intersections of \( N_u \), the set of neighbors of \( u \), with the sets of the neighbors of each neighbor of \( u \) (Lines 4-7).**

```plaintext
1 /* Input: A graph. Output: Triangle count tc \in N. */
2 //Derive a vertex order R s.t. if R(v) < R(u) then dv <= du
3 for v \in V [in par] do:
4 \quad N_v^+ = \{ u \in N_v | R(v) < R(u) \}
5 tc = 0; //Init tc, for all neighbor pairs, increase tc.
6 //Now, derive the actual count of triangles:
7 for u \in N_v^+ [in par] do:
8 \quad N_u^+ = \{ v \in N_u | R(u) < R(v) \}
9 \quad tc += |N_u^+ \cap N_v^+|

Listing 1: Triangle Counting (Node Iterator).
```
We also consider higher-order Clique Counting, a problem important for dense subgraph discovery [68]. Listing 2 contains 4-clique listing. We reformulated the original scheme [68] (without changing its time complexity) to expose \(|X \cap Y|\). The algorithm presented generalizes TC.

Listing 2: Reformulated 4-Clique Counting.

```
1 /* Input: A graph G. Output: Number of 4-cliques c_k \in \mathbb{N}. */
2 /* Derive a vertex order R s.t. \( R(v) < R(u) \) then \( d_v \leq d_u \): */
3 for \( v \in V \) [in par] do: \( N^+_v = \{ u \in N_v | R(v) \leq R(u) \} \)
4 \( c_k = 0 \);
5 for \( u \in V \) do: \( N^+_u \cap N^+_v \) /*Find 3-cliques*/
6 for \( w \in N^+_u \cap N^+_v \) [in par] do:
7 \( C_3 = N^+_u \cap N^+_v \) /*Find 3-cliques*/
8 for \( w \in C_3 \) do: /*For each 3-clique...*/
9 \( c_k += \{ N^+_u \cap C_3 \} \) /*Find 4-cliques*/
```

Vertex Similarity measures, used in graph databases and others [69]–[73], assess how similar two vertices \( v \) and \( u \) are, see Listing 3. They can be used on their own, or as a building block of more complex algorithms such as clustering. Many of these schemes use the cardinality of set intersection. This includes Jaccard, Common Neighbors, Total Neighbors, or Adamic Adar. Vertex Similarity is the basic building block of Link Prediction and Clustering.

Graph Clustering is a broadly studied problem [42]. Listing 4 shows Jarvis-Patrick clustering [44], a scheme that uses vertex similarity to determine whether these two vertices are in the same cluster, and relies heavily on \(|X \cap Y|\).

Listing 4: Jarvis-Patrick clustering.

```
1 /* Input: A graph G = \( (V, E) \). Output: Clustering C \subseteq V. */
2 /* of a given prediction scheme. */
3 /* Use a similarity \( SC(v, u) = |N_v \cap N_u| \) (see Listing 3). */
4 for \( e = (v, u) \in E \) [in par] do: /* \( \tau \) is a user-defined threshold*/
5 if \( |N_v \cap N_u| > \tau \) then: \( C \cup= (e) \)
6 /* Other clustering schemes use other similarity measures. */
```

```
1 /* Input: A graph G = \( (V, E) \). Output: Similarity S \in \mathbb{R} of sets \( A, B \). */
2 /* Most often, \( A \) and \( B \) are neighborhoods \( N_u \) and \( N_v \), */
3 /* of vertices \( u \) and \( w \). */
4 /* Jaccard similarity: */
5 \( S_J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \)
6 /* Overlap similarity: */
7 \( S_O(A, B) = \frac{|A \cap B|}{\min(|A|, |B|)} \)
8 /* Certain measures are only defined for neighborhoods: */
9 \( S_A(v, w) = \frac{\sum_{u \in N_v} (1 / \log |N_u|)}{\text{where } w \in N_v \cap N_u} \), Adamic Adar
10 \( S_P(v, w) = \frac{\sum_{u \in N_v} (1 / |N_u|)}{\text{where } w \in N_v \cap N_u} \), Resource Alloc.
11 \( S_C(v, w) = |N_v \cap N_u| \), Common Neighbors
12 \( S_T(v, w) = |N_v \cup N_u| = |N_v| + |N_u| - |N_v \cap N_u| \), Total Neighbors
```

Graph Clustering is a widely studied problem [42]. Listing 4 shows Jarvis-Patrick clustering [44], a scheme that uses vertex similarity to determine whether these two vertices are in the same cluster, and relies heavily on \(|X \cap Y|\).

Link Prediction There are many schemes for predicting whether two non-adjacent vertices can become connected in the future in the context of evolving networks [75]. Assessing the accuracy of a specific link prediction scheme \( \Delta \) is done with a simple algorithm [76] shown in Listing 5. We start with some graph with known links (edges). We derive \( \Delta_{\text{sparse}} \subseteq E \), which is \( E \) with random links removed; \( \Delta_{\text{sparse}} = E \setminus \Delta_{\text{rndm}} \). \( \Delta_{\text{rndm}} \subseteq E \) are randomly selected missing links from \( E \) (links to be predicted). We have \( \Delta_{\text{sparse}} \cup \Delta_{\text{rndm}} = \emptyset \) and \( \Delta_{\text{sparse}} \cap \Delta_{\text{rndm}} = \emptyset \). Now, we apply the link prediction scheme \( \Delta \) (that we want to test) to each edge \( e \in (V \times V) \setminus \Delta_{\text{sparse}} \). The higher a value \( \Delta(e) \), the more probable \( e \) is to appear in the future (according to \( \Delta \)). Now, the effectiveness of \( \Delta \) is computed by verifying how many of the edges with highest prediction scores \( \Delta_{\text{scores}} \) actually are present in the original dataset \( E \): \( ef = |\Delta_{\text{scores}} \cap \Delta_{\text{rndm}}| \).
Listing 5: Link prediction testing.

A. Real-World Applications

Graph problems targeted by ProbGraph have numerous real-world applications because the underlying operation $|X \cap Y|$, used to find the counts of the shared neighbors, is a common building block in many real-world problems in domains ranging from network science or sociology, through chemistry or biology, to the Internet studies [42].

Triangle counting is used to obtain the network cohesion, an important measure of connectedness and “togetherness” of a group of vertices [77], [78]. Specifically, for any subgraph $S \subseteq V$, $S$’s cohesion is $TC(S) / (|S|^3)$, where $TC(S)$ is the triangle count of $S$; note that $S$ may also form $V$ (in which case we obtain the cohesion of the whole graph). Another example is discovering communities [79], [80], by computing the clustering coefficient defined as $3 \cdot TC(S) / (|S|^3)$. Other use cases include spam detection (standard and spam sites differ in the respective counts of triangles that they belong to), optimization of query planning in databases [81], uncovering hidden thematic layers in WWW [82], or studying differences between gene interactomes of various species [83].

The considered Jarvis-Patrick clustering can be used in adaptive web search based on automatic construction of user profiles. A critical step in this use case is generation of clusters of users, which is directly achieved using the clustering scheme addressed in PG [84]. Other selected examples are drug design (by predicting plasma protein bindings [85]), screening and generating overviews of chemical databases (by computing clusters of related molecules) [86], or analyzing single-cell RNA sequences (by approximating smooth low-dimensional surfaces that model states of cells) [87], [88].

Other considered problems also have numerous applications. In short, clique counting is used in social network analysis (cf. the established textbooks [89, Chapter 11] and [90, Chapter 2]) to find large and dense network regions [91]–[95] or in topological approaches to network analysis [96]. Link prediction and vertex similarity are used throughout the whole graph data mining in many parts of network science and others, as illustrated in numerous surveys and textbooks [4], [11], [12], [75], [97]–[99].

IV. APPROXIMATING $|X \cap Y|$ |X \cap Y|

We now show how to derive approximate set intersection cardinality $|X \cap Y|$ both fast and with high accuracy. We provide selected results for BF and MH (less competitive outcomes are in the Appendix). In this section, we assume arbitrary sets $X$ and $Y$, to ensure that our outcomes are of interest beyond graph mining. From Section VII onwards, we focus on graph mining by applying the results from this section to $|N_u \cap N_v|$.

A. Section Overview and Intuition

We first outline the section structure and provide the intuition behind the key parts. We first present estimators for $|X \cap Y|$, designed using BF (§ IV-B), $k$-Hash (§ IV-C), and 1-Hash (§ IV-D). Then, we compare the obtained estimators regarding their accuracy (§ IV-E) and the amount of parallelism (§ IV-F).

We provide concentration bounds for all the estimators. We present here selected ones, the others are in the supplementary material together with the proofs of all the propositions of this section. A generic form of a concentration bound from this section is $P(\text{estimator} - \text{true value} \geq t) \leq f(t)$. Intuitively, this means that we can bound (i.e., by the function $f(t)$) the probability that a PG estimator deviates (i.e., more than $t$) from the true parameter value for every sketch size. The function $f(t)$ (either polynomial or exponential in $t$ for all PG estimators) determines the speed at which a given PG estimator concentrates around the true parameter value.

B. Approximating $|X \cap Y|$ with Bloom Filters

We introduce a new estimator $|X \cap Y|_{\text{AND}}$ and we give a bound on its accuracy. Specifically, for two sets $X$ and $Y$ represented by $B_X$ and $B_Y$, we apply the estimator from Eq. (1) to $B_X \cap Y$, obtaining

$$|X \cap Y|_{\text{AND}} = - \frac{B_{X \cap Y}}{b} \log \left(1 - \frac{B_{X \cap Y_1}}{B_{X \cap Y}}\right)$$

(2)

where $B_{X \cap Y} = B_X = B_Y$ is the BF size (cf. Table I). Next, we prove an important property of $|X \cap Y|_{\text{AND}}$. Note that the following property also holds for the estimator by Swamidass [59] from Eq. (1).

Proposition IV.1. Let $|X \cap Y|_{\text{AND}}$ be the estimator defined in Eq. (2). For $B_{X \cap Y} \geq \log \left(1 + o(1)\right)\left(e^{X \cap Y} |X \cap Y|_{\text{AND}} \geq 0.499B_{X \cap Y} \cdot \log B_{X \cap Y}\right)$ the following holds:

$$E \left[\left(|X \cap Y|_{\text{AND}} - |X \cap Y|\right)^2\right] \leq (1 + o(1)) \left(e^{X \cap Y} |B_{X \cap Y} - \log B_{X \cap Y}| \right)$$

Overall, Proposition IV.1 shows that we can bound the mean squared error (MSE) of $|X \cap Y|_{\text{AND}}$ (and also $|X|_S$ from Eq. (1)) by Chebyshev’s inequality, we obtain the following concentration result:

$$P \left(|X \cap Y|_{\text{AND}} - |X \cap Y| \geq t\right) \text{ rather than } P \left(|X \cap Y|_{\text{AND}} - E(|X \cap Y|_{\text{AND}}) \geq t\right) (\text{as is usually done).}$$
The probability of deviation from the true intersection set size, at a given distance \( t \geq 0 \), is:

\[
P \left( \left| \hat{X} \cap \hat{Y} \right|_{kH} - \left| X \cap Y \right| \geq t \right) \leq 2e^{- \frac{b_k t^2}{\left(\left| X \cap Y \right| + t\right)^2}}
\]  

(6)

We stress that \( \left| \hat{X} \cap \hat{Y} \right|_{kH} \) derived with k-hash can also be interpreted as a maximum likelihood estimator (MLE) (cf. § II-E) for \( |X \cap Y| \) because of the invariance property outlined in § II-F (details are provided together with the proof). Thus, our estimator inherits all the properties of MLE such as consistency and asymptotic efficiency. Moreover, the bound is exponential, i.e., the distance between the estimator and the true value of \( |X \cap Y| \) decreases exponentially. Finally, we stress that \( \left| \hat{X} \cap \hat{Y} \right|_{kH} \) is asymptotically efficient, i.e., no other estimator (using only this sketch) can have lower variance (for a fixed input and asymptotically for \( k \to \infty \)).

D. Approximating \( |X \cap Y| \) with 1-Hash

The 1-Hash estimator is similar to k-Hash in that we first estimate the Jaccard similarity itself as \( \hat{J}_{X,Y}^{1H} = \frac{|M_X \cap M_Y|}{K} \). Similarly to the estimator used in k-Hash, this is itself an unbiased estimator of \( J_{X,Y} \). Then, we use it to estimate \( X \cap Y \): \( \left| \hat{X} \cap \hat{Y} \right|_{1H} = \frac{\hat{J}_{X,Y}^{1H}}{1 + \hat{J}_{X,Y}^{1H}} (|X| + |Y|) \).

Recall that the 1-Hash representation of \( X \) differs qualitatively from the k-Hash variant in that (1) \( M_X \) does not contain duplicates, and (2) \( M_X \) uses only one hash function. The \( k \) elements maintained in a 1-Hash are not independent, as we are in a sampling without replacement scheme4. This also means that k-Hash can have duplicates, which is not possible with 1-Hash. Thus, \( |M_X \cap M_Y| \sim \text{Hypergeometric}(|X \cup Y|, |X \cap Y|, k) \), where \( |X \cup Y| \) is the population size, \( |X \cap Y| \) is the number of success states in the population, and \( k \) is the number of draws. This implies that we can derive the expectation and the variance of \( \left| X \cap Y \right|_{1H} \) by adapting the formulas for the moments of an hypergeometric random variable. We provide the formulas for the expectation in the Appendix. We now provide the same concentration bound as in the case of k-Hash.

**Proposition IV.3.** Consider \( \left| \hat{X} \cap \hat{Y} \right|_{1H} \). Then, an upper bound for the probability of deviation from the true intersection set size, at a given distance \( t \geq 0 \), is:

\[
P \left( \left| \hat{X} \cap \hat{Y} \right|_{1H} - \left| X \cap Y \right| \geq t \right) \leq 2e^{- \frac{b_1 t^2}{\left(\left| X \cap Y \right| + t\right)^2}}
\]  

(7)

The bound suggests that 1-Hash can be better in practice than k-Hash. They both have exponential bounds but 1-Hash requires hashing elements using only one hash function. Thus, it is faster to compute.

E. Analysis of Accuracy of \( |X \cap Y| \)

We summarize our theory developments into estimating \( |X \cap Y| \) in Table II (estimators) and Table III (bounds). These results are also applicable to general estimators of \( |X| \)

---

3To satisfy this assumption, one could just store perfectly random permutations on the set of vertices without increasing asymptotic complexity.

4Contrarily, k-Hash is a sampling with replacement scheme and explains why \( |M_X \cap M_Y| \sim \text{Bin}(k, J_{X,Y}) \) for k-Hash.
We provide deviation bounds for all PG estimators. The estimator for $k$-hash is an MLE. Moreover, the $k$-Hash and 1-Hash bounds are exponential. This means that the estimates are unlikely to deviate much from the true value.

<table>
<thead>
<tr>
<th>Result</th>
<th>Where</th>
<th>Class</th>
<th>AU</th>
<th>CN</th>
<th>ML</th>
<th>IN</th>
<th>AE</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>X</td>
<td>_2$</td>
<td>Eq. (1)</td>
<td>BF</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>$</td>
<td>X \cap Y</td>
<td>_{AND}$</td>
<td>Eq. (2)</td>
<td>BF</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>$</td>
<td>X \cap Y</td>
<td>_I$</td>
<td>§ IV-B</td>
<td>BF</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>$</td>
<td>X \cap Y</td>
<td>_H$</td>
<td>§ IV-D</td>
<td>1-Hash</td>
<td>✗</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

TABLE II: Summary of theoretical results (estimators) related to $|X|$ and $|X \cap Y|$. “★” a new result provided in this work (a new estimator or proving a certain novel property of a given estimator). “CN”: a consistent estimator. “AU”: an asymptotically unbiased estimator. “ML”: an MLE estimator. “IN”: an invariant estimator. “AE”: an asymptotically efficient estimator.

<table>
<thead>
<tr>
<th>Result</th>
<th>Where</th>
<th>Class</th>
<th>Q</th>
<th>MS</th>
<th>CO</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>X</td>
<td>_2$</td>
<td>Eq. (1)</td>
<td>BF</td>
<td>P</td>
</tr>
<tr>
<td>$</td>
<td>X \cap Y</td>
<td>_{AND}$</td>
<td>Eq. (3)</td>
<td>BF</td>
<td>P</td>
</tr>
<tr>
<td>$</td>
<td>X \cap Y</td>
<td>_I$</td>
<td>§ IV-B</td>
<td>BF</td>
<td>P</td>
</tr>
<tr>
<td>$</td>
<td>X \cap Y</td>
<td>_H$</td>
<td>§ IV-D</td>
<td>1-Hash</td>
<td>E</td>
</tr>
</tbody>
</table>


F. Analysis of Parallelism in $|X \cap Y|$

In Table IV, we provide a work-depth analysis of parallelism in different estimators, when applied to intersecting vertex neighborhoods. For the exact intersection applied to different estimators, when applied to intersecting neighborhoods, when applied to intersecting vertex neighborhoods (BF, $k$-Hash, or 1-Hash), (2) plugs in PG routines for obtaining $|X \cap Y|$ in place of the exact set

<table>
<thead>
<tr>
<th>CSR (merge)</th>
<th>CSR (gallop.)</th>
<th>BF</th>
<th>$k$-Hash</th>
<th>1-Hash</th>
</tr>
</thead>
<tbody>
<tr>
<td>Work: $O(d_u + d_v)$</td>
<td>$O(d_u \log d_u)$</td>
<td>$O \left( \frac{d_u}{W} \right)$</td>
<td>$O(k)$</td>
<td>$O(k)$</td>
</tr>
<tr>
<td>Depth: $O(\log(d_u + d_v))$</td>
<td>$O(\log(d_u + d_v))$</td>
<td>$O \left( \frac{d_v}{W} \right)$</td>
<td>$O(\log k)$</td>
<td>$O(\log k)$</td>
</tr>
</tbody>
</table>

TABLE IV: Work and depth of simple parallel algorithms for deriving $|N_u \cap N_v|$ (cardinality of the result of intersecting neighborhoods of vertices $u, v$).

intersections. For example, to use PG with graph algorithms from Section III, one replaces the operations indicated with blue color with PG routines. As an example, in Listing 6, we compare obtaining Jaccard similarity of two neighborhoods with an exact scheme and with a PG routine based on BF. We ensure that one can flexibly select an arbitrary estimator $|X \cap Y|$ because – as our evaluation (Section VIII) shows – no single representation works best in all cases.

A. Tradeoffs Between Storage, Accuracy, & Performance

Each probabilistic set representation considered in PG offers a tradeoff between performance, storage, and accuracy. In general, the smaller a representation is, the faster to process it becomes and the less storage it needs, but also the less accurate it becomes. To control this tradeoff, we introduce a generic parameter $s$ that enables explicit control of the storage budget.

B. USING PROBGRAPH WITH GRAPH ALGORITHMS

We carefully design and implement PG as an easy-to-use library offering different set representations. To use PG, the user (1) creates selected probabilistic representations of vertex neighborhoods (BF, $k$-Hash, or 1-Hash), (2) plugs in PG routines for obtaining $|X \cap Y|$ in place of the exact set

V. DESIGN & IMPLEMENTATION

Each BF is implemented as a simple bit vector, $|X \cap Y|$ can then be computed using bitwise AND over $X$ and $Y$, with Eq. (2). Computing $B_{X \cap Y}$ can easily be parallelized and accelerated with vectorization [102]: the problem is embarrassingly parallel and the bitwise AND is supported with SIMD technologies such as AVX deployed in Intel CPUs, GPUs, and others. We also use the popcnt CPU instructions [103] to speed up deriving the number of ones in a bit vector (1-bits), needed to obtain $B_{X \cap Y,1}$ in Eq. (2); popcnt counts the number of 1-bits in one memory word in one CPU cycle.

1-Hash and $k$-Hash are both series of integers. The estimators for $|X \cap Y|$ based on 1-Hash and $k$-Hash are dominated by intersecting sets of $k$ numbers. As $k \ll d$, it is much faster than the corresponding operations on exact neighborhoods.
A. Parallel Construction

Table V provides work and depth of constructing all probabilistic PG set representations used in PG. As with the intersection computation, the construction process is also parallelizable, exhibiting very low depth. During evaluation (Section VIII), we show that it also does not pose a bottleneck in practice.

<table>
<thead>
<tr>
<th>Representation of $N_v$</th>
<th>Size (bits)</th>
<th>Construction (work)</th>
<th>Construction (depth)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BF</td>
<td>$B_X$</td>
<td>$O(bd_v)$</td>
<td>$O(</td>
</tr>
<tr>
<td>$k$-Hash</td>
<td>$W_k$</td>
<td>$O(kd_v)$</td>
<td>$O(d_v)$</td>
</tr>
<tr>
<td>1-Hash</td>
<td>$W_k$</td>
<td>$O(d_v)$</td>
<td>$O(log(d_v))$</td>
</tr>
</tbody>
</table>

TABLE V: Work/depth of simple algorithms for constructing a probabilistic PG set representation of a given neighborhood $N_v$. In BF, one must iterate over all $k$ hash functions and all $d_v$ neighbors, thus the work is dominated by $bd_v$ (cf. § II-D). All the hash function evaluations can run in parallel, but – in the worse case – they may write to the same cell in the BF bit vector, giving depth $O(|log(bd_v)|)$ (parallelization with a binary tree reduction). Derivations for MH are similar; the work and depth are dominated by evaluations of hash functions and by finding $k$ smallest elements among $d_v$ ones, respectively.

B. Parallelism in ProbGraph-Enhanced Graph Algorithms

Parallelization of graph algorithms enhanced with ProbGraph is straightforward and is based on the listings from Section III. Specifically, all the loops marked with [in par] can be executed in parallel. Then, all the instances of set intersection cardinality are executed using a user-specified PG estimator. The parallel execution of these estimators (cf. § IV-F and Table IV) enables better work and depth of graph mining algorithms than with the default CSR implementation. We illustrate this in Table VI. Here, work and depth of CSR based routines are standard results known from extensive works in parallel algorithm design [7], [9], [56], [65], [104]. For example, in TC, the two outermost loops can be executed fully in parallel, and the nested set intersection dominates depth ($d$ is the maximum degree in a graph). Both work and depth for PG baselines are derived by replacing the nested exact $|X \cap Y|$ operation with the corresponding PG schemes and results from Table IV. These asymptotic advantages are supported with empirical outcomes detailed in Section VIII.

C. Implementation Details and Infrastructure

We use the GMS platform [4], a high-performance parallel graph mining infrastructure, for implementing the baselines. Loading graphs from disk and building the CSR representations is done with the GAP Benchmark Suite [105]. We use the MurmurHash3 hash function [106], well-known for its speed and simplicity. We use the current time in milliseconds as a random seed. For parallelization, we use OpenMP [107]. The whole implementation is available online.5

VII. THEORETICAL ANALYSIS OF ACCURACY

We now illustrate that ProbGraph enables obtaining not only strong theoretical accuracy guarantees on the set intersection cardinality, but also on graph properties. As an example, we now use our estimators $|X \cap Y|$ to develop estimators $\tilde{TC}$ for triangle count $TC$, and to derive its concentration bounds.

As shown in Listing 1, TC can be obtained by summing intersections $|N_u \cap N_v|$ of neighborhoods for each pair of adjacent vertices $u$ and $v$. Hence, to estimate TC, we simply sum cardinalities $|N_u \cap N_v|$ for each edge $(u, v)$ in a given graph. This gives the following estimator:

$$\tilde{TC}_e = \frac{1}{3} \sum_{(u, v) \in E} |N_u \cap N_v|,$$

where * indicates a specific $|X \cap Y|_*$ estimator (cf. Table II).

Theorem VII.1. Let $\tilde{TC}_e$ be the estimator of the number of triangles. (cf. Section III). Then, depending on the underlying estimator $|X \cap Y|_*$, we have the following cases:

For the Bloom Filter AND estimator, if $b \Delta \leq 0.499B_X \log B_X$, then we have the following bound

$$P \left( \left| TC - \tilde{TC}_{AND} \right| \geq t \right) \leq \frac{2 m^2 (1 + o(1)) \left( \frac{\pi^{\frac{B_X}{B_Y}} B_Y}{\Delta} - \frac{B_X}{B_Y} - \frac{\Delta}{t} \right)}{9 t^2}.$$

In the case of both 1-Hash and $k$-Hash (below, we use the notation for 1-Hash), we have

$$P \left( \left| TC - \tilde{TC}_{1H} \right| \geq t \right) \leq 2 \exp \left( -\frac{18 k t^2}{(\sum_{v \in V} d(v))^2} \right).$$

Moreover, if the maximum degree is $\Delta$, then

$$P \left( \left| TC - \tilde{TC}_{1H} \right| \geq t \right) \leq 2 \exp \left( -\frac{9 k t^2}{4 (\Delta + 1) \sum_{v \in V} d(v)^3} \right).$$

We provide a detailed proof of each statement of Theorem VII.1 in the Appendix.

Consistency of all the TC estimators follows from consistency of the individual estimators (cf. § II-F). The fact that $\tilde{TC}_{kH}$ is MLE follows from $|X \cap Y|_{kH}$ being MLE.

A. Comparison to Existing Estimators

We compare our $\tilde{TC}$ estimators to others in Table VII. We consider the estimators from Doulion [46], topological graph sketching [48], GAP [50], ASAP [49], Slim Graph [51], MCMC [108], and the “colorful” TC analysis [47], as well as

<table>
<thead>
<tr>
<th></th>
<th>CSR</th>
<th>PG (BF)</th>
<th>PG (MH)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triangle Counting (work):</td>
<td>$O(nd^2)$</td>
<td>$O\left(\frac{ndB_X}{W}\right)$</td>
<td>$O(ndk)$</td>
</tr>
<tr>
<td>Triangle Counting (depth):</td>
<td>$O(\log d)$</td>
<td>$O\left(\frac{log B_X}{W}\right)$</td>
<td>$O(\log k)$</td>
</tr>
<tr>
<td>4-Clique Counting (work):</td>
<td>$O(nd^2)$</td>
<td>$O\left(\frac{ndB_X}{W}\right)$</td>
<td>$O(ndk)$</td>
</tr>
<tr>
<td>4-Clique Counting (depth):</td>
<td>$O(\log^2 d)$</td>
<td>$O\left(d \log \left(\frac{B_X}{W}\right)\right)$</td>
<td>$O(\log^2 k)$</td>
</tr>
<tr>
<td>Clustering (work):</td>
<td>$O(nd^2)$</td>
<td>$O\left(\frac{ndB_X}{W}\right)$</td>
<td>$O(ndk)$</td>
</tr>
<tr>
<td>Clustering (depth):</td>
<td>$O(\log d)$</td>
<td>$O\left(\frac{log B_X}{W}\right)$</td>
<td>$O(\log k)$</td>
</tr>
<tr>
<td>Vertex sim. (work):</td>
<td>$O(d^2)$</td>
<td>$O\left(\frac{B_X}{W}\right)$</td>
<td>$O(k)$</td>
</tr>
<tr>
<td>Vertex sim. (depth):</td>
<td>$O(\log d)$</td>
<td>$O\left(\frac{log B_X}{W}\right)$</td>
<td>$O(\log k)$</td>
</tr>
</tbody>
</table>

TABLE VI: Advantages of ProbGraph in work and depth over exact baselines.
A. Datasets, Methodology, Architectures

This means that – for both original work does not explicitly provide a given result and it was derived in (§ II-F). “AC” (asymptotically unbiased), “CN” (consistent), “ML” (maximum likelihood estimator), “IN” (invariant), “AE” (asymptotically efficient): properties of estimators (explained in § II-F). “P” (polynomial) or “E” (exponential): bound quality, “CY” supported, provided. “X” not available, not provided. “i”: the original work does not explicitly provide a given result and it was derived in this work. Symbols used in related work (different from ones in Table I): p: probability of keeping an edge, m: #sampled edges.

We also consider state-of-the-art heuristics that do not come with theoretical guarantees: Reduced Execution [112], Partial Processing [112], Auto-Approximate (two variants) [113].

Datasets We consider SNAP (S) [114], KONECT (K) [115], DIMACS (D) [116], Network Repository (N) [117], and WebGraph (W) [118] datasets. For broad analysis, we follow the recommendations of the GMS graph mining benchmark [4], and we use networks of different origins (biology, chemistry, economy, etc.), sizes, densities (m/n), degree distribution skews, and even higher-order characteristics (e.g., counts of cliques). We illustrate the real-world datasets in Table VIII. We also use synthetic graphs power-law (the Kronecker model [119]) degree distribution. Using such synthetic graphs enables systematically changing a specific single graph property such as n, m, or m/n, which is not possible with real-world datasets. This entails a very large evaluation space and we only include representative findings for selected graphs.


Brain. (bh-medical, 8K, 8.9K), (bh-mouse, 1.1K, 90.8K), (bh-mouse_brain_f, 213, 21.8K).

Economic. (ecom-pimgrid, 3.1K, 543K), (ecom-pimgrid2, 3.1K, 540K), (ecom-beaHs, 498, 50.4K), (ecom-beas, 508, 54K), (ecom-nb-mac), 493, 49.9K), (ecom-oranet78, 2.5K, 90.1K).

Social. Facebook: (soc-FBMsg, 1.9K, 13.8K), Orkut: (3.1M, 117M).

Scientific computing. (sc-pkrt, 217.9K, 5.6M), (sc-OptGupt, 16.8K, 4.7M), (sc-Therm&M, 10.6K, 522.4K).


Chemistry. (Chem-SiO, 33.4K, 675.5K), (Chem-SiO1H16, 1.7K, 446.5K).

TABLE VIII: ProbGraph vs. existing results for estimating TC (sorted chronologically). “Constr. time”: time to construct a given estimator. “Memory”: the amount of storage needed to construct a given estimator. “Estimation time”: time needed to estimate TC. “AC” (asymptotically unbiased), “CN” (consistent), “ML” (maximum likelihood estimator), “IN” (invariant). “AE” (asymptotically efficient): properties of estimators (explained in § II-F). “BF” (concentration bounds): whether a given scheme is supported with concentration bounds. “P” (polynomial) or “E” (exponential): bound quality. “CY”: supported, provided. “X”: not available, not provided. “i”: the original work does not explicitly provide a given result and it was derived in this work. Symbols used in related work (different from ones in Table I): p: probability of keeping an edge, m: #sampled edges.

We follow a recent set of recommendations on the benchmarking parallel applications [109]. For example, we omit the first 1% of performance data as warmup. We derive enough data for the mean and 95% non-parametric confidence intervals.

Comparison Baselines We compare PB-based approximate algorithms to tuned state-of-the-art implementations (Triangle and 4-Clique Counting, Clustering, and Vertex Similarity) from the GAP [105] and GMS [4] graph benchmarking suites. Moreover, when analyzing our estimators of \(|N_u \cap N_v|\), we consider an existing BF estimator [110], [111], given by the expression \( |X| = -\log(1 - b_1 \times b_2 / b_3) \). We also consider another existing estimator [59], given by the expression \( |X \cap Y|_{OR} = \left| X \right| + \left| Y \right| - \frac{\left| X \right| \times \left| Y \right|}{\left| X \cap Y \right|_{OR}} \). This estimator uses the single set estimator evaluated on the set union. Finally, when evaluating TC, we compare to the established TC estimators: Douillon [46] (representing schemes based on edge sampling) colorfull TC [47] (representing schemes based on sophisticated combinatorial pruning).

We use the generic Eden et al. [52] and Doulion [46] with concentration bounds.

<table>
<thead>
<tr>
<th>Reference</th>
<th>Constr. time</th>
<th>Memory used</th>
<th>Estimation time</th>
<th>A</th>
<th>U</th>
<th>CN</th>
<th>ML</th>
<th>IN</th>
<th>AE</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Doulion [46]</td>
<td>O(m)</td>
<td>O(pm)</td>
<td>O(T(pm))</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Colorful [47]</td>
<td>O(m)</td>
<td>O(pm)</td>
<td>O(T(pm))</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Sketching [48]</td>
<td>O(kn^2)</td>
<td>O(kn)</td>
<td>O(T(k^n))</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>ASAP [49]</td>
<td>O(n)</td>
<td>O(n + m)</td>
<td>O(1)/sample</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>GAP [50]</td>
<td>O(m)</td>
<td>O(m^{1/2})</td>
<td>O(T(m^{1/2}))</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Slim Gr. [51]</td>
<td>O(m)</td>
<td>O(pm)</td>
<td>O(T(pm))</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Eden et al. [52]</td>
<td>n/a</td>
<td>O(T(\sqrt{m}))</td>
<td>O(T(\sqrt{m}) + \frac{\sqrt{m}}{2})</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Assaad et al. [53]</td>
<td>n/a</td>
<td>O(km)</td>
<td>O(m^{3/2}/TC)</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Techk [54]</td>
<td>n/a</td>
<td>O(km)</td>
<td>O(m^{3/2}/TC)</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
</tbody>
</table>

TABLE VII: ProbGraph vs. existing results for estimating TC (sorted chronologically). “Constr. time”: time to construct a given estimator. “Memory”: the amount of storage needed to construct a given estimator. “Estimation time”: time needed to estimate TC. “AC” (asymptotically unbiased), “CN” (consistent), “ML” (maximum likelihood estimator), “IN” (invariant). “AE” (asymptotically efficient): properties of estimators (explained in § II-F). “BF” (concentration bounds): whether a given scheme is supported with concentration bounds. “P” (polynomial) or “E” (exponential): bound quality. “CY”: supported, provided. “X”: not available, not provided. “i”: the original work does not explicitly provide a given result and it was derived in this work. Symbols used in related work (different from ones in Table I): p: probability of keeping an edge, m: #sampled edges.
Similarly, comparably to marginally better than L on sparser graphs. On the other hand, MH usually gives consistently higher speedups as well as lower memory requirements, but its accuracy is in most cases worse than BF. We conjecture this is because MH estimators preserve only specific subsets of vertex neighborhoods (selected using hash functions), explicitly eliminating other vertices. In contrast, when using BF estimators, each vertex is hashed to certain bit(s) in the final bit vector, and is thus to some extent “reflected” in PG estimators.

In terms of memory efficiency, the mostly very light shades indicate very low additional storage overheads. Except for a few cases where light gray indicates that – for a given graph

Cray supercomputer (one such node comes with 12-core Intel Xeon E5-2690 HT-enabled CPU 64 GiB RAM).

**Parallelism** Unless stated otherwise, we run algorithms on the maximum number of cores available in a system.

**Assessing Accuracy** To measure the accuracy of algorithms that return some counts (e.g., clique count, count of clusters), we use expression $\left|\mathit{cnt}_{\mathit{PG}} - \mathit{cnt}_{\mathit{EX}}\right|$, where $\mathit{cnt}_{\mathit{PG}}$ and $\mathit{cnt}_{\mathit{EX}}$ are ProbGraph and exact counts, respectively. Note that the ProbGraph counts may be lower but also higher than the exact ones (due to false positives in BFs).

**B. Estimating $|N_v \cap N_u|$**

We first assess specific PG estimators of $|N_v \cap N_u|$ in terms of their accuracy. For each graph, we derive the BF and MH representations of its vertex neighborhoods, and then the intersections of neighborhoods of adjacent vertices. Finally, we compute the relative differences between these PG intersection cardinalities and the CSR related cardinalities $|\mathit{PG} \cap \mathit{CSR}_{\mathit{EX}}|/|\mathit{PG} \cap \mathit{CSR}_{\mathit{EX}}|$ where $\bullet \in \{\text{AND, L, 1H, kH}\}$. We summarize these differences, for each graph, using boxplots. We use the storage budget $s = 33\%$ and $b \in \{1, 4\}$.

Representative results are in Figure 3. While medians are low (less than $\approx 25\%$ for most cases), there is a certain spread in outliers. This is because we consider all adjacent vertices, and there is a high chance that at least some pairs will result in low accuracy. Overall, the results illustrate that there is no single winner among the estimators, and the outcomes depend on the graph structure. One observation is that the BF based on AND tends to perform worse on very dense graphs, and comparatively to marginally better than L on sparser graphs. Similarly, $k$-Hash is marginally worse than 1-Hash on very dense graphs; sparse graphs entail a rare pattern.

**C. Estimating Outcomes of Graph Algorithms**

We now illustrate that PG estimators enable high performance and high accuracy at a small additional storage budget when applied to parallel graph mining. We first conduct an analysis using all 32 cores. For each graph problem considered, we illustrate the exact baseline and the schemes based on BF and MH estimators. We use $|N_u \cap N_v|_{\text{AND}}$ with $b = 2$ and $|N_u \cap N_v|_{1H}$ that represent BF and MH schemes; they offer high accuracy while being fast to compute as they need few hash functions (other estimators come with similar accuracy outcomes but are slower to compute). Figures 4 and 5 show the results for real-world and Kronecker graphs. Each single plot is dedicated to a specific graph problem and it compares different estimators (indicated with different shapes of data points) across three dimensions: performance (speedup, X axis), accuracy (relative count, Y axis), and memory budget (relative memory size with respect to the default CSR, shades of B&W). Each plot corresponds to a specific graph problem. Each data point corresponds to the execution of a given scheme for a specific graph dataset. Thus, each plot shows collectively how different baselines behave for different input graphs.

In general, the results follow the insights from the analysis of estimating set intersections. BFs offer high accuracy and high speedups, sometimes even as high as $20 \times$ (Clustering using Overlap), or nearly $30 \times$ (Clustering using Common Neighbors), while keeping the accuracy more than 98%. Speedups can be as high as $50 \times$, with the accuracy more than 90% (e.g., 4-Clique Counting for Kronecker graphs). On the other hand, MH usually gives consistently higher speedups as well as lower memory requirements, but its accuracy is in most cases worse than BF. We conjecture this is because MH estimators preserve only specific subsets of vertex neighborhoods (selected using hash functions), explicitly eliminating other vertices. In contrast, when using BF estimators, each vertex is hashed to certain bit(s) in the final bit vector, and is thus to some extent “reflected” in PG estimators.

In terms of memory efficiency, the mostly very light shades indicate very low additional storage overheads. Except for a few cases where light gray indicates that – for a given graph...
Each single data point corresponds to the execution of a given scheme for a single specific graph dataset.

---

**D. Comparison to Heuristics**

We conclude that BF-based PG estimators consistently deliver high accuracy as well as speedups at small memory budget, for a broad set of graphs and problems. 1–Hash based schemes may provide much more performance, but require more careful parametrization and input selection.

Moreover, we use two variants of sampling-based “Auto-Approximation” that addresses a purely vertex-centric model of computation [113]. We show these heuristics in Figure 6 (we exclude them from Figures 4 and 5 to preserve clarity; they always achieve much worse results that obscure plots). The advantage of heuristics is that they do not need additional memory, as shown in the lowest panel in Figure 6. However, PG always achieves much better accuracy, by at least 25%, up to $\approx 75\%$. This is because the heuristics are not based on theoretical developments that ensure high PG’s accuracy.
Moreover, the heuristics are also slower than PG. “AutoApproximate” schemes introduce particularly large overheads due to their purely vertex-centric abstraction, which makes them even slower than the exact tuned baselines that we compare to. The results for all other graph problems are similar.

E. Analysis of Scaling

We also consider strong/weak scaling; Figure 8 offers representative results. We observe nearly ideal strong scaling of all the baselines compared. PG schemes feature much lower runtimes. To investigate in what regime of parameters PG of all the baselines compared. PG schemes feature much lower representative results. We observe nearly ideal strong scaling.

E. Analysis of Scaling

We also consider strong/weak scaling; Figure 8 offers representative results. We observe nearly ideal strong scaling of all the baselines compared. PG schemes feature much lower runtimes. To investigate in what regime of parameters PG of all the baselines compared. PG schemes feature much lower representative results. We observe nearly ideal strong scaling.

E. Analysis of Scaling

We also consider strong/weak scaling; Figure 8 offers representative results. We observe nearly ideal strong scaling of all the baselines compared. PG schemes feature much lower runtimes. To investigate in what regime of parameters PG of all the baselines compared. PG schemes feature much lower representative results. We observe nearly ideal strong scaling.

IX. Beyond Bloom Filter and MinHash

The generic nature of PG enables using other probabilistic representations in place of BF and MH. As an example, we discuss how to use PG with K Minimum Values (K MV) [123], another sketch that was originally developed to accelerate counting distinct elements in a data stream. To construct a KMV representation \( K_X \) of a set \( X \), one evaluates the associated hash function \( h : X \rightarrow \{0; 1\} \) for all elements of \( X \). Then, one selects \( k \) smallest hashes that constitute the final KMV representation \( K_X \) of the set \( X \). One can then estimate \( |X| \) with \( |X|_{\text{KMV}} = \frac{k}{\max_{X,Y} |K_X \cap K_Y|} \). Note that \( K_X \) differs from a MH \( M_X \) because, as opposed to \( M_X \), it contains hashes.

Now, one can use KMV to also estimate \( |X \cap Y| \), and then use it within PG. For this, one constructs a KMV \( K_{X \cap Y} \) by taking the \( k \) smallest elements from \( K_X \cup K_Y \). Then, by the KMV properties, we have \( |X \cup Y|_{\text{KMV}} = \frac{k-1}{\max_{X,Y} |K_X \cup K_Y|} \). Finally, \( |X \cap Y|_{\text{KMV}} = |X| + |Y| - |X \cup Y|_{\text{KMV}} \), which can be directly used in PG formulations of graph algorithms.

We provide concentration bounds for all the KMV estimators defined above in the Appendix.

X. Related Work: Summary

We summarize related work; some parts are already covered in Sections I and VII. First, there exist more set-related probabilistic data structures, for example HyperLogLog [124]. ProbGraph embraces such data structures: while we focus on BF [35] and MH [37], one could easily extend ProbGraph with other structures; we leave details for future work.

Next, there are many approximate graph algorithms [20]–[24], [26]–[30], [30]–[33], [49]. ProbGraph differs from them as it can approximate any algorithm or scheme that uses \( |X \cap Y| \), set membership query, and others, where \( X \) and \( Y \) are arbitrary sets of vertices or edges (all our theoretical and most of empirical results are directly applicable to any sets). Moreover, ProbGraph is simple: all one has to do is to plug in a selected set representation.

A few existing general approaches for approximate graph computations usually target specific problems or they do not come with guarantees on the quality of outcomes [108], [112], [113]. As shown in Section VII, unlike ProbGraph, specific schemes for TC do not offer strong accuracy guarantees [46]–[51].

ProbGraph’s probabilistic representations of vertex sets are a form of graph compression [125], and they could be used to extend existing compressed graph representations and paradigms [51], [126].
There exist a few works on using BF or MH specific single graph problems, usually in the context of evolving graphs [48], [127]–[130], which is outside PG’s scope.

Approximating the triangle count in time less then linear in the size of the input was shown in [52]. This has been later generalized to approximating the number of $k$-cliques in a graph [131]. Improved bounds are known when the arboricity of the graph is small [132]. Assuming we can sample edges uniformly, better algorithms are also known [53], [133]. Unlike PG, these schemes are specific to selected graph problems and graphs with certain properties such as low arboricity.

There are many works on counting or finding different graph patterns (also called motifs, graphlets, or subgraphs) [1], [3], [4], [9], [12], [14], [68], [74], [75], [97], [134]–[147]. PG can be used as a subroutine in different such works, offering speedups while preserving high accuracy.

Counting and listing simple patterns such as triangles have been recently used to enhance the design of numerous models in Graph Neural Networks [148], [149], [149]–[151], [151], [152], [152], [153], [153], [154], [154]–[159]. Such models could use PG to accelerate expensive graph mining preprocessing costs.

The straightforward parallelism in computing BF based estimators implies that other architectures that offer massive parallelism may provide even higher benefits. This includes FPGAs [27], [160]–[162], CGRAs [163], or processing in-memory [164]–[172]. We leave these studies for future work.

Next, there are many approximate graph algorithms [18], [22], [173], [173]–[176]. ProbGraph differs from them as it is general: it can approximate any algorithm or scheme that uses $|X \cap Y|$, set membership query, and others, where $X$ and $Y$ are arbitrary sets of vertices or edges (all our theoretical and most of empirical results are directly applicable to any sets). Moreover, ProbGraph is simple: all one has to do is to plug in a selected set representation and implementations of $|X \cap Y|$, a set membership query, and any other related schemes.

**XI. CONCLUSION**

We propose ProbGraph, a parallel graph representation that enables simple, general, and high-performance approximate graph computations. The key idea is to sketch sets of vertices, and the cardinality of the intersection of such sets, with probabilistic set representations such as Bloom filters or MinHash. Such representations usually offer much higher performance than exact set representations, while only requiring small additional storage. Importantly, they can be treated as a black box and seamlessly incorporated into graph algorithms. We show that ProbGraph is simple to use while offering speedups of more than 50× for some graphs and retaining high accuracy of more than 90% for problems such as Triangle Counting, when comparing to tuned exact parallel baselines on 32 cores.
(a) Strong scaling (TC).
(b) Strong scaling (Clustering, Common Neighbors Vertex Similarity).
(c) Strong scaling (Clustering, Jacard Vertex Similarity).
(d) Strong scaling (Clustering, Overlap Vertex Similarity).
(e) Weak scaling (TC).
(f) Weak scaling (Clustering, Common Neighbors Vertex Similarity).
(g) Weak scaling (Clustering, Jacard Vertex Similarity).
(h) Weak scaling (Clustering, Overlap Vertex Similarity).

Fig. 8: Analysis of scaling of representative baselines.

Fig. 9: Scaling results for Clustering (Common Neighbors), illustrating comparable scaling performance of both BF and MH.

We support ProbGraph with an in-depth theoretical underpinning, in which we derive novel statistical concentration bounds on the accuracy of ProbGraph approximations. Our bounds are the first exponential or polynomial quality bounds on the accuracy of ProbGraph approximations. Our pinning, in which we derive novel statistical concentration bounds are of interest to the broader audience beyond graph mining on processing-in-memory systems, “arXiv preprint arXiv:2103.03653, 2021.

**Acknowledgements:** We thank Hussein Harake, Colin McMurry, Mark Klein, Angelo Mangili, and the whole CSCS team granting access to the Ault and Daint machines, and for their excellent technical support. We thank Timo Schneider for immense help with computing infrastructure at SPCL. We thank Aryaman Fasciati and Sebastian Leisinger for help in the early stages of the project. This research received funding from the European Research Council (Project DAPP, No. 678880; Project PSAP, No. 101002047), Huawei, and MS Research through its Swiss Joint Research Centre. Jakub Tćtek was supported by the VILLUM Foundation grant 16582.

**References**


Here, we provide proofs and various details omitted from the main part of the manuscript.

A. Probability Distributions

Consider a sequence of $n$ trials (experiments). A single trial is a Bernoulli trial, i.e., it gives a success or a failure outcome with a probability $p$ or $1-p$, respectively. In the context of PG, a single trial will correspond to some property of a given set representation, for example whether a given bit in a BF has 1 or 0. Now, if all the trials are independent (i.e., obtaining a specific outcome does not impact the number of such potential outcomes in future trials), the resulting distribution is binomial (commonly denoted as $Bin(n,p)$). Otherwise (i.e., obtaining a specific outcome decreases by one the number of such potential outcomes in future trials), it is hypergeometric (usually denoted as $Hyper(N,K,n)$).

Both distributions enable deriving specific probabilities for the number of either success or failure trials.

B. Plug-In Principle

Some concentration bounds that we present in this paper depend on a given set size (i.e., $|X|$ appears in the formulation of the bound). Thus if we want to obtain an estimate of the upper bound, we need to substitute the estimator $|X|$ instead of $|X|$ whenever $|X|$ appears. This procedure is known as the plug-in principle and it is well established in statistics.

However this method is safe to use only if the estimator we substitute is at least consistent for the parameter of interest (i.e., $|X|$). Indeed this is the case for all the BF and MinHash estimators presented in this paper.

C. BF Sketches for Single Sets

We provide extended results for BF for single sets.

1) Concentration Bound for BF Single Sets:

**Proposition A.1.** Let $\hat{|X|}_b$ be the estimator defined in Eq. (1). For $B_X, b \in \mathbb{N}$ such that $b = o(\sqrt{B_X})$, and a set $X$ such that $b |X| \leq 0.499 B_X \log B_X$ the following holds:

$$E \left[ \left( \hat{|X|}_b - |X| \right)^2 \right] \leq (1+o(1)) \left( e^{\frac{X^{1/6}}{B_X}} \frac{B_X}{b^2} - \frac{B_X}{b^2} - |X| \right)$$

**Proof.** We now prove Proposition A.1. Before bounding the mean squared error of $\hat{|X|}_b$, we need to prove several simple bounds. Let $\mu = E[B_{0,X}] = B_X \left( 1 - \frac{1}{B_X} \right)^{b|X|}$. It holds:

$$\mu \geq B_X \left( 1 - \frac{1}{B_X} \right)^{0.499B_X \log B_X} \geq B_X \exp \left( -\frac{0.499 \log B_X}{1 - \frac{1}{B_X}} \right) = B_X^{0.501 - o(1)}$$

**APPENDIX**
Let us fix some $\varepsilon > 0$. Let $\mathcal{E}$ be the event that $B_{0,X} \geq \mu/(1+\varepsilon)$. [177, Theorem 2] prove that:

$$P(\mathcal{E}) \geq 1 - \exp(-\Omega(\mu^2/B_X)) \geq 1 - \exp\left(-B_X^4(11)\right).$$

We have $|X| = -\frac{b_X}{2} \log(B_{X,0}/B_X + 1| B_{X,0} = 0) \leq B_X \log B_X$ and by our assumption, $|X| \leq b|X| \leq 0.499B_X \log B_X$. It thus holds $(|X| - |X|)^2 \leq O(B_X^2 \log^2 B_X)$. Let $\kappa = -\frac{B_X}{2} \log \left(1 - \frac{1}{4\pi_X}\right) = -B_X|X| \log \left(1 - \frac{1}{4\pi_X}\right)$. Moreover, for $B_X \to \infty$, we have $\log(1 - 1/B_X) = -1/B_X + O(1/B_X^2)$. Therefore, it holds $\kappa = |X| + o(1)$.

Now we are able to bound the mean squared error of $|X|$. We present each step of the derivation as a unique figure (see Fig. 10 below) to improve the clarity of the content.

In particular, eq. (10) in Figure 10 holds because for any $a, b, c \in \mathbb{R}$ and $\varepsilon > 0$, it holds $|a - b|^2 \leq |1 + \varepsilon|^2|a - c|^2 + \frac{|a - c|^2}{(1+\varepsilon)^2}(c - b)^2$. Eq. (13) holds because on $\mathcal{E}$, given $B_{X,0} \geq \mu/(1+\varepsilon)$, $\log(B_{X,0}/B_X)$ is $c$-lipschitz for $c = (1+\varepsilon)B_X/\mu \leq (1+\varepsilon)e^{b|X|/2\pi_X} = (1+\varepsilon)e^{\frac{b|X|}{2\pi_X}}$. Eq. (16) holds because $E[B_{X,0}/B_X] = (1 - 1/B_X)^b|X|$ and eq. (17) holds because $Var(B_{X,0}/B_X) \sim B_X e^{-\frac{b|X|}{2\pi_X}} - B_X \left(\frac{b|X|}{2\pi_X} + 1\right) e^{-\frac{b|X|}{2\pi_X}}$. By sending $\varepsilon \to 0$, we get that:

$$E[(|X| - |X|)^2] \leq (1 + o(1))\left(e^{-\frac{|X|b}{2\pi_X}} B_X \alpha \beta - B_X \beta^2 |X| \beta \right)$$.}

2) Class of Estimators with General Bounds: The bound on the MSE presented in Proposition A.1 holds up to some assumptions (i.e. $b = o(\sqrt{B_X})$ and $b|X| \leq 0.499B_X \log B_X$) and an $o(1)$ term. To derive a concentration bound for the MSE which does not depend on these conditions and that enhance the interpretability, we develop a class of estimators which encompasses the one by Swamidass et al. [59]. To introduce this framework, we first propose a new limiting estimator called $|X|_L$ which belongs to this class. We obtain $|X|_L$ by simplifying the estimator from Eq. (1) and taking the limit for $B_X \to \infty$:

$$|X|_L \equiv \lim_{B_X \to \infty} \left|\hat{X}_S \right| = \lim_{B_X \to \infty} \frac{B_X}{b} \log \left(1 - \frac{B_{X,1}}{B_X} \right) = \log \left(\lim_{B_X \to \infty} \left(1 - \frac{B_{X,1}}{B_X} \right) \right) = \log \left(\exp \left(\frac{B_{X,1}}{b} \right) \right) = \frac{B_{X,1}}{b}. \quad (20)$$

We can perform this simplification thanks to the continuity of the logarithm in $(0, \infty)$ that allows us to safely move the limit inside log, knowing that $B_{X,b} \in \mathbb{N}$ by construction. This result tells us that, as $B_X$ increases, $|X|_S$ rescales the number of ones in the BF by the quantity $\frac{1}{b}$ because $|X|_S \sim \frac{B_{X,1}}{B_X}$ for $X, b$ fixed and $B_X \to \infty$. We can also prove that $|X|_S \leq \log \frac{B_{X,1}}{B_X}$, thus implying that $|X|_S$ can inflate the number of ones at most by the factor $\log \frac{B_{X,1}}{B_X}$. These interesting insights motivate us to propose a general class of estimators. The key idea is to define any BF estimator as a function of a random variable (i.e. $B_{X,1}$, the number of ones in a BF). Specifically, we have $|X|_L \equiv \delta_{X,b}(B_{X,1})$, where $\delta(\cdot)$ is a given non-negative function of $B_{X,1}$, and $B_{X,1}$. We choose to denote $\delta_{B_{X,1},B_X}$ and $\delta_{B_{X,1}}$ instead of the usual $\delta(B_{X,1})$ and $\delta_{X,b}$, B_{X,1} to clearly separate the deterministic BF design parameters $B_X$ and $b$ from the unique random component $B_{X,1}$. The key benefit of this formulation is that (1) it generalizes both $|X|_S$ and $|X|_L$, and (2) we can use it to provide concentration bounds that are applicable to $|X|_L$ and many other estimators within the proposed class depending on the functional form of $\delta_{B_{X,1},B_X}$.

To obtain $|X|_S$, we set:

$$|X|_S \equiv \delta_{B_{X,1},B_X} = B_X \frac{B_{X,1}}{B_X} \log \left(1 - \frac{B_{X,1}}{B_X} \right).$$

To recover $|X|_L$, we first (with a slight abuse of notation) fix $\delta_{B_{X,1},B_X}$ to be linear in $B_{X,1}$ and then set it to be specifically equal to $\frac{1}{b}$:

$$|X|_L \equiv \delta_{B_{X,1}}(B_{X,1}) = \frac{B_{X,1}}{B_X}. \quad (21)$$

We underline that if $\delta_{B_{X,1}}(B_{X,1})$ is linear in $B_{X,1}$ we are implicitly imposing, depending on the values of $B_X$ and $b$, either a deflation or an inflation of the observed number of ones in the BF. For example, we have already seen that, when $B_X \to \infty$ for fixed $X, b$, we have $\delta_{B_{X,1}}(B_{X,1}) = \frac{b}{b}$ in Eq. (1). We now show that any estimator that can be written as $|X|_L$ with $\delta_{B_{X,1}}(B_{X,1})$ linear in $B_{X,1}$ has a bounded MSE.

**Proposition A.2.** Let $|X|_L \equiv \delta_{B_{X,1}}(B_{X,1})$. For $B_X, b \in \mathbb{N}$, the following holds:

$$E\left[\left(|X|_L - |X|\right)^2\right] \leq \left|X| - \delta_{B_{X,1}}(B_X) \left(1 - e^{-|X|b/2\pi_X}\right)\right|^2 + \delta_{B_{X,1}}^2(B_X) e^{-|X|b/2\pi_X} \left(1 + \frac{|X|b}{2\pi_X}\right) e^{2|X|b/2\pi_X}.$$
\[
E[(\tilde{X} - |X|)^2] = E[(\tilde{X} - |X|)^2]\mathcal{E}P(\mathcal{E}) + E[(\tilde{X} - |X|)^2] - \mathcal{E}P(\mathcal{E}) \leq (1 + \varepsilon)E[(\tilde{X} - \kappa)^2\mathcal{E}] + \frac{1 + \varepsilon}{\varepsilon}E[(\kappa - |X|)^2\mathcal{E}] + O(B_X^2 \log^2 B_X \cdot \exp(-B_X^2))
\]

which ends the proof. To improve the interpretability of the bound, we use the fact that \((1 - \frac{1}{\pi_X})\frac{b}{b} \sim e^{-\frac{X}{\pi_X}}\) in the statement of Proposition A.2.

3) Enhancing the Estimator by Swamidass [59]:

The estimator by Swamidass et al. [59], is divergent in its original form. To alleviate this, we replace \(B_{X,1} = B_{X,1} - \|B_{X,1} - B_X\|\), where, for a given proposition \(P\), \(\|P\| = 1\) if \(P\) holds, and 0 otherwise. \(\tilde{B}_{X,1}\) only differs from \(B_{X,1}\) in the unlikely case of \(B_{X,1} = B_X\). Thanks to this modification, our estimator \(|\tilde{X}|\) has, unlike Swamidass et al.’s, a finite expectation (as it is bounded).

The final form of the estimator is

\[
\tilde{X} = -\frac{B_X}{b} \log \left(1 - \frac{\tilde{B}_{X,1}}{B_X}\right)
\]

4) Proof of consistency and asymptotic unbiasedness:

We need to show that \(|\tilde{X}| = -\frac{B_X}{b} \log \left(1 - \frac{\tilde{B}_{X,1}}{B_X}\right)\) is consistent and asymptotically unbiased as \(B_X \to \infty\). We provide here an intuitive formulation based on the false positive probability which can be easily made more rigorous by direct application of the definition of consistency which we omit for the sake of simplicity. First of all, as shown in eq.(21), we can notice that
\[ |X|_S \sim |X|_L \] as the Bloom Filter size diverges. This means that the proof is valid for both estimators because they are asymptotically equivalent. Now we can look at the probability of false positives as \( B_X \to \infty \) for fixed and finite \( b \) and \( |X| 
\]

\[
\lim_{B_X \to \infty} \left[ 1 - \left( 1 - \frac{1}{B_X} \right)^{b|X|} \right]^b \to 0
\]

The result above tells us that false positive matches cannot happen anymore in the limit. Each element of \( |X| \) will then be hashed in a \textit{personal} bit and counting the number of ones in \( B_X \) (and dividing by \( b \) in case of multiple hash functions) will always deliver \( |X| \) at a given precision as \( |X| \) is fixed and \( B_X \to \infty \). Thus we can conclude that \( \frac{B_X}{b} \cdot \frac{b}{b} |X| \) which proves consistency. Asymptotic unbiasedness follows from consistency in our case as the variance of both estimators is bounded (see the proof of Proposition IV.1). The same reasoning can be easily extended to show consistency and asymptotic unbiasedness also for \( |X \cap Y|_{AND} \) and \( |X \cap Y|_{OR} \) presented in section IV-B.

D. Proposition IV.1

\textbf{Proof.} To prove Proposition IV.1 from Section IV-B we can extend in a straightforward way the proof presented for Proposition A.1. Indeed we just need to substitute \( |X| \) with \( |X \cap Y| \) and \( B_X \) with \( B_X \cap Y \) to obtain the desired result. \( \square \)

E. MinHash Sketches for Set Intersection

1) Expectation formula: Since in the case of k-hash, \( |M_X \cap M_Y| \sim Bin( k, J_{X,Y} ) \), and for 1-hash, \( |M_X \cap M_Y| \sim Hypergeometric(|X \cup Y|, |X \cap Y|, k) \), we have:

\[
E[|X \cap Y|_{kH}] = (|X| + |Y|) \sum_{s=0}^{k} \binom{k}{s} (J_{X,Y})^s (1 - J_{X,Y})^{k-s} \frac{s}{k + s} \tag{23}
\]

\[
E[|X \cap Y|_{1H}] = (|X| + |Y|) \sum_{s=0}^{k} \binom{|X \cup Y|}{s} \binom{|X \cap Y|}{k-s} \frac{s}{k + s} \tag{24}
\]

There exists an involved closed form expression for equation (23) which is beyond the scope of this paper. We refer the interested reader to [179] for a clear derivation of a similar problem.

2) Proof of consistency and asymptotic unbiasedness: We start to show that \( |X \cap Y|_{kH} \) is consistent. This follows respectively from Proposition IV.2 statement. Indeed by taking the limit for \( k \to \infty \) with fixed and finite \( |X| \) and \( |Y| \) we obtain:

\[
\lim_{k \to \infty} P \left( |X \cap Y|_{kH} - |X \cap Y| \geq t \right) \leq \lim_{k \to \infty} 2e^{-\frac{2k s^2}{(|X \cup Y|)^2}} \to 0
\]

The above implies that \( |X \cap Y|_{kH} \) \( \Rightarrow \) \( |X \cap Y| \). On the other hand, for \( |X \cap Y|_{1H} \) we are in the \textit{sampling without replacement} scheme. This means that the population size (i.e. \( |X \cup Y| \)) is finite and by taking the limit for \( k \to |X \cup Y| \) in Proposition IV.3, with fixed and finite \( |X| \) and \( |Y| \), we have already sampled the entire population contrarily to the k-Hash case. Thus \( |X \cap Y|_{1H} \) is also a consistent estimator of \( |X \cap Y| \). Then, for both estimators, the asymptotic unbiasedness follows from consistency and by noticing that both \( |X \cap Y|_{kH} \) and \( |X \cap Y|_{1H} \) have a bounded variance.

3) Sub-Gaussian preliminaries: We recall some key notions of sub-gaussian random variables as they are necessary for the following proofs. First of all, we define \( \psi_X(\lambda) = \log(\mathbb{E}[e^{\lambda X}]) \) as the logarithmic moment generating function (i.e. cumulant) of a generic random variable \( X \). For example, if \( Z \) is a centered normal random variable with variance \( \sigma^2 \), we have that \( \psi_Z(\lambda) = \frac{\lambda^2 \sigma^2}{2} \). It can be shown, we refer the interested reader to chapter 2 of [38] for a detailed explanation, that Chernoff’s inequality in this case implies, for all \( t > 0 \), that:

\[
P( Z \geq t ) \leq e^{-\frac{\sigma^2 t^2}{2}} \tag{25}
\]

The bound above, characterize the decay of the tail probabilities of a centered normal random variable. If the tail probabilities of a generic centered random variable \( X \), decrease at least as rapidly as the ones in (25) then \( X \) is \textit{sub-gaussian}. More formally, a centered random variable \( X \) is said to be \textit{sub-gaussian} with variance factor \( \sigma^2 \) if:

\[
\psi_X(\lambda) \leq \frac{\lambda^2 \sigma^2}{2} \quad \forall \lambda \in \mathbb{R} \tag{26}
\]

We underline that (26) only requires \( \text{Var}(X) \leq \sigma^2 \). Moreover, if we call \( G(\sigma^2) \) the collection of random variables for which (26) holds (e.g. all bounded random variables belongs to \( G(\sigma^2) \)), we can state that:

\textbf{Lemma A.3.} Let \( X_1, \ldots, X_n \) be \textit{sub-gaussian} random variables so that \( X_i \in G(\sigma_i^2) \) for every \( i \in \{ 1, \ldots, n \} \). Then \( \sum_{i=1}^{n} X_i \in G((\sum_{i=1}^{n} \sigma_i^2)^{2}) \). Moreover if the random variables are independent, then \( \sum_{i=1}^{n} X_i \in G((\sum_{i=1}^{n} \sigma_i^2)^{2}) \).

This is due to the fact that (26) implies a bound on the moment generating function whose properties, together with the Hölder inequality, help to verify the above statement. For a detailed proof of lemma A.3 see theorem 2.7 in [?] while for alternative characterizations of sub-gaussianity in terms of growth of moments, we refer to chapter 2 of [38].

4) Concentration bounds for k-Hash and 1-Hash: We present below the proof of Propositions IV.2 and IV.3. First, we show the following lemma which we will also use later.

\textbf{Lemma A.4.}

\[
P(|\hat{J}_1 - J| \geq t), P(|\hat{J}_k - J| \geq t) \leq 2e^{-\frac{kt^2}{(1+|X \cup Y|)^2}} \tag{27}
\]

\textbf{Proof.} The random variables \( k, \hat{J}_1 \) and \( k, \hat{J}_k \) follow the hypergeometric and binomial distributions, respectively. Applying the Hoeffding’s inequalities in the binomial case, we get the desired inequality. The Serfling’s bound can be applied in the case of the hypergeometric distribution. The Serfling’s bound always gives better bounds than the Hoeffding’s, proving the inequality for \( \hat{J}_1 \). \( \square \)
We now show concentration of the sum of the estimators and, therefore also of the individual estimators (by fixing \( n = 1 \)).

**Theorem A.5.** Let \( Y_1 = \sum^n_i C_i \frac{j_i}{1+j_i}, Y_k = \sum^n_i C_i \frac{j_k}{1+j_k}. \) Then for any non-negative constants \( C_i \) and \( S = \sum^n_i C_i \frac{j}{1+j} \),

\[
P(\vert Y_1 - S \vert > t), P(\vert Y_k - S \vert > t) \leq 2 \exp \left( -\frac{2 k t^2}{(\sum^n_i C_i )^2} \right)
\]

(28)

**Proof.** The function \( X \) is 1-Lipschitz and it, therefore, holds that \( \forall X, X' \in [0,1] \)

\[
\left| \frac{X}{1+X} - \frac{X'}{1+X'} \right| \leq |X - X'|
\]

The concentration result from Lemma A.4 then also holds for \( \frac{j_i}{1+j_i} \) and \( \frac{j_k}{1+j_k} \). The random variables

\[
\frac{j_i}{1+j_i} - \frac{J_i}{1+J_i}, \frac{j_k}{1+j_k} - \frac{J_k}{1+J_k}
\]

are therefore sub-gaussian with variance factor \( \sigma^2 = \frac{1}{4k} \) (see eq. (25), (26) and in general section E3). Now we multiply by \( C_i \) each variable, we take the sum and, thanks to lemma A.3, we get that \( Y_1 - S \) and \( Y_k - S \) are sub-gaussian with variance factor

\[
\frac{(\sum^n_i C_i )^2}{4k}
\]

The theorem follows from the definition of a sub-gaussian random variable (see section E3 for further references).

We stress here that \( |X \cap Y|_{kH} \) derived with \( k \)-hash can also be interpreted as a maximum likelihood estimator (MLE) (cf. § II-E) for \( |X \cap Y| \) because of the invariance property outlined in § II-F and detailed in Chapter 7 of [39]. Indeed since \( |M_X \cap M_Y| \sim Bin(k, J_{X,Y}) \) we have that \( J_{X,Y}_{kH} = |M_X \cap M_Y| / k \) is the maximum likelihood estimator of \( J_{X,Y} \) if we assume that the \( k \) hash functions are independent and perfectly random (a usual assumption). Then our estimator \( |X \cap Y|_{kH} \) is just a function of \( J_{X,Y}_{kH} \) and, because of the invariance of the MLE (see Theorem 7.2.10 in [39]), this implies that \( |X \cap Y|_{kH} \) inherits all the properties of this class of estimators. In particular, it is consistent and asymptotically efficient since it reaches the Cramer-Rao Lower Bound (see theorem 7.3.9 in [39]) meaning that no other estimator can have a lower variance. It is also normally distributed, as \( k \) increase, which is useful in general to derive confidence intervals.

**F. Results & Derivations for Triangle Counts**

1) **Proof of consistency and asymptotic unbiasedness:** Any estimator for triangle count analyzed in PG, is simply a sum of cardinalities \( |X \cap Y| \) for different neighborhoods \( X \) and \( Y \) (cf. Section III):

\[
\bar{TC} = \frac{1}{3} \sum_{(u,v) \in E} |N_u \cap N_v|_\star
\]

where \( \star \) indicates a specific \( |X \cap Y| \) estimator (cf. Table II).

Since we have already proven consistency and asymptotic unbiasedness for each of the \( |X \cap Y| \) estimators presented in PG, we can now address jointly the consistency of the triangle count estimators. To do so we just need to acknowledge the fact that a sum of consistent estimators is itself a consistent estimator. Indeed this is a direct consequence of the more general Slutsky theorem which enable us to state that \( \bar{TC} \xrightarrow{p} TC \). The asymptotic unbiasedness then follows from consistency and by noticing that all \( \bar{TC} \) estimators have a bounded variance (see all the proofs presented below).

2) **Bloom Filters:** We first present the estimator \( |X \cap Y|_{OR} \). This estimator was introduced before [59] and uses the single set estimator evaluated on the set union:

\[
|X \cap Y|_{OR} = |X| + |Y| + \frac{B_{X \cup Y}}{B_{X \cup Y}} \log \left( 1 - \frac{B_{X \cup Y}}{B_{X \cup Y}} \right)
\]

(29)

Note that, to obtain the expression above, we also use the fact that \( |X \cup Y| = |X| + |Y| - |X \cap Y| \).

We now prove the triangle count bound for BF stated in theorem VII.1 for the triangle count OR estimator (the proof is of course valid also for \( \bar{TC}_{AND} \)).

**Proof.** We first define the mean squared error (mse) as follows:

\[
E[(|TC - \bar{TC}_{OR}|^2)] = E[\bar{TC}_{OR} - TC]^2 + Var(\bar{TC}_{OR})
\]

where the equality is a standard identity.

Now we bound the first component of the mse which is the squared bias of our estimator. In order to ease the notation from now on we denote \( |N_u \cup N_v| \forall (u,v) \in E \) as \( |X| \forall i = 1, \ldots, m \) where \( m \) is the number of edges. In the same fashion, we denote \( |X| \forall i = 1, \ldots, m \) as the estimator of \( |N_u \cup N_v| \forall (u,v) \in E \). Thus we can write:

\[
E[(|TC_{OR} - TC|)^2]
\]

(30)

\[
\leq \frac{1}{9} \left\{ \sum_{i=1}^m \sum_{j=1}^m \left[ E[|X_i|] - |X_i| \right] \left[ E[|X_j|] - |X_j| \right] \right\}
\]

(31)

\[
\leq \frac{1}{9} \left\{ \sum_{i=1}^m \sum_{j=1}^m \left[ E[|X_i|] - |X_i| \right] \left[ E[|X_j|] - |X_j| \right] \right\}
\]

(32)

\[
= \frac{1}{9} \left\{ \sum_{i=1}^m \sum_{j=1}^m \sqrt{E[|X_i|^2] - |X_i|^2} \sqrt{E[|X_j|^2] - |X_j|^2} \right\}
\]

(33)

\[
\leq \frac{m^2}{9} \left( 1 + o(1) \right) \left( e^{2\Delta b/(B_X - 1)} \frac{B_X}{b^2} - \frac{B_X}{b^2} - \frac{2\Delta}{b} \right)
\]

(34)
where (32) follows by Cauchy–Schwarz inequality and (34) by Proposition A.1 which in general bounds the mse (and thus the squared bias by Jensen’s inequality) if $2\Delta \leq 0.499B_X \log B_X$ where $B_X = \min(B_{N_u \cap N_v})$ with $(u,v) \in E$. In particular we underline that any bound obtained by Proposition A.1, which is valid for a given set size, is also automatically valid for all the set sizes smaller than that a fortiori. Thus we can notice that $2\Delta \geq |X_i| \forall i = 1, \ldots , m$ where $\Delta$ is the maximum degree of the input graph which justifies (34). At this point, it remains to bound the second component of the mse which is the variance of our estimator. Indeed we can write:

$$\text{Var}(TC_{OR}) = \frac{1}{9} \text{Var} \left[ \sum_{i=1}^{m} |X_i| \right]$$

(35)

$$= \frac{1}{9} \sum_{i=1}^{m} \sum_{j=1}^{m} \text{Cov}(|X_i|, |X_j|)$$

(36)

$$\leq \frac{1}{9} \sum_{i=1}^{m} \sum_{j=1}^{m} \sqrt{\text{Var}(|X_i|)} \sqrt{\text{Var}(|X_j|)}$$

(37)

$$\leq \frac{m^2}{9} \left( 1 + o(1) \right) \left( e^{2\Delta b/(B_X-1)} \frac{B_X}{b^2} - \frac{B_X}{b^2} - \frac{2\Delta}{b} \right)$$

(38)

where (37) holds because of the covariance inequality and (38) by Proposition A.1 which in general bounds the mse (and thus the variance a fortiori) if $2\Delta \leq 0.499B_X \log B_X$. The justification of (38) is similar to the one outlined before for the squared bias case.

Now, again assuming $2\Delta \leq 0.499B_X \log B_X$, we can obtain the bound for the mean squared error of the OR estimator:

$$E[(TC - \overline{TC}_{OR})^2] =$$

$$= (E[TC_{OR}] - TC)^2 + \text{Var}(\overline{TC}_{OR})$$

$$\leq \frac{2m^2}{9} \left( 1 + o(1) \right) \left( e^{2\Delta b/(B_X-1)} \frac{B_X}{b^2} - \frac{B_X}{b^2} - \frac{2\Delta}{b} \right)$$

The above bound is valid also for $\overline{TC}_{AND}$ since $\Delta \geq |N_u \cap N_v| \forall (u,v) \in E$ however it can be made tighter for the same reason. Indeed if $b\Delta \leq 0.499B_X \log B_X$ (where now $B_X = \min(B_{N_u \cap N_v})$ with $(u,v) \in E$) by Chebychev inequality:

$$P \left( |TC - \overline{TC}_{AND}| \geq t \right) \leq$$

$$\frac{2m^2}{9} \left( 1 + o(1) \right) \left( e^{\Delta b/(B_X-1)} \frac{B_X}{b^2} - \frac{B_X}{b^2} - \frac{2\Delta}{b} \right)$$

which is the statement of Theorem VII.1 for the BF case.

3) MinHash: We can show the concentration of the sum of the set intersection estimators using theorem A.5 presented in Appendix E. Then for the edge $e_i = uv$, we define $C_i = \deg(u) + \deg(v)$ thus giving us $S = \frac{1}{3} \sum_{i=1}^{m} C_i \frac{j}{1+j}$.

$TC$. We will not consider the scaling factor $\frac{1}{3}$ till the final expressions of the bounds to ease the notation. Thus we can write:

$$\sum_{i=1}^{m} C_i = \sum_{i=1}^{m} \deg(u) + \deg(v)$$

$$= \sum_{v \in V} \deg(v)^2$$

Combining the above result with theorem A.5, we obtain the triangle count bound for MinHash presented in Theorem VII.1.

However this bound can be improved if we assume more independence, which will be satisfied in the case of triangle counting when the maximum degree is not too large. We now prove a tighter bound under these conditions.

**Theorem A.6.** Let $Y_1 = \sum_{i=1}^{m} C_i \frac{j}{1+j}$, $Y_k = \sum_{i=1}^{m} C_i \frac{j}{1+j}$, and assume we partition the set of estimators into groups $X_1, \ldots , X_k$ such that estimators from each set are mutually independent. Then for any non-negative constants $C_i$ and $S = \sum_{i=1}^{m} C_i \frac{j}{1+j}$

$$P(|Y_1 - S| > t), P(|Y_k - S| > t) \leq 2 \exp \left( - \frac{k \max(0, t - 2S/k)^2}{2 \left( \sum_{i=1}^{m} C_i \frac{j}{1+j} \right)^2} \right)$$

(39)

Proof. We modify the proof of Theorem A.5 by instead considering the random variables

$$\frac{J}{1+J} - \frac{J}{1+J} - \frac{\mu_1}{1+J}$$

$$\frac{J}{1+J} - \frac{J}{1+J} - \frac{\mu_k}{1+J}$$

where $\mu_1$ and $\mu_k$ are chosen so as to make this random variable have mean zero.

We then first sum estimators from each group separately using Lemma A.3, which gives us subgaussian coefficient (i.e. the square root of the variance factor) of $\sqrt{\sum_{x=\chi} C_x^2}$. Adding the groups together, we get using again Lemma A.3, that the subgaussian coefficient is $\sigma_X = \sum_{x=\chi} \sqrt{\sum_{d=\chi} C_d^2}$. To finish the proof of the first inequality, we have to show a bound on $\sum C_i \mu_1$ and $\sum C_i \mu_k$. We show the argument for the case of 1-hash, the argument for $k$-hash is analogous. Note that $\mu_1$ is the Jensen gap of $\frac{J}{1+J}$, Var $J$ $\leq J/k$, by Theorem 1 from [180], we have $-J/k \leq E[\frac{J}{1+J}] - \frac{J}{1+J} \leq 0$. We can bound $-J/k \geq 2/k \frac{j}{1+j}$. Therefore, we can bound

$$-2S/k \leq \sum C_i \mu_1 \leq 0$$

To prove the second inequality, we define the following optimization problem

maximize $\sum_{i=1}^{n} \sqrt{x_i}$

subject to $\sum x_i = c$
Set $x_i = \sum_{d \in X_i} C_d^2$ and $c = \sum C_t^2$. We see that for every possible assignment of the estimators to the sets $\{X_i\}_{i=1}^X$, we have a feasible solution with objective value equal to the subgaussian coefficient $\sigma_X$. Therefore, the subgaussian coefficient for any assignment to the groups is dominated by the optimum of this optimization problem.

Optimum of this optimization problem is when all $x_i$’s have the same value – otherwise one can pick $i, j$ such that $x_i < x_j$ and $0 < \varepsilon \leq (x_j - x_i)/2$ and then replace $x_i$ by $x_i + \varepsilon$ and similarly $x_j$ by $x_j - \varepsilon$, increasing the objective while retaining feasibility. This gives us objective value of $\chi \sqrt{\sum_{i=1}^m C_i^2}/\chi = \sqrt{\chi \sum_i C_i^2} / \chi$.

To show the final expression of the bound, we use Theorem A.6. Then by Vizing’s theorem, $\chi \leq \Delta + 1$ and by the same substitution done for the first bound, we have:

$$
\sum_{i=1}^m C_i^2 = \sum_{i=1}^m (\deg(u) + \deg(v))^2 \leq \sum_{i=1}^m 2(\deg(u)^2 + \deg(v)^2) = 2 \sum_{v \in V} \deg(v)^3
$$

Indeed combining the above result with Theorem A.6, we obtain the triangle count bound for MinHash presented in Theorem VII.1 if the maximum degree is $\Delta$.

G. KMV Sketches

1) Single Sets: We state an existing result on the KMV sketching; we use it later to provide a KMV sketch for $|X \cap Y|$ [181]. The hash function used with a KMV maps elements from $X$ to real numbers in $(0, 1]$ u.a.r.9. Thus, the hashes should be evenly spaced and one can estimate $|X|$ by dividing the size $k - 1$ of $K_X$ by the largest hash in $K_X$.

$$
|\hat{X}|_K = \frac{k - 1}{\max \{x | x \in K_X\}}
$$

(39)

As noted in [181, §2.1], the k-th smallest value follows the beta distribution $Beta(\alpha, \beta)$ with shape parameters $\alpha = k$ and $\beta = |X| - k + 1$. Now we can get concentration bounds for the estimator: indeed, following [182], we can show that:

**Proposition A.7.** Consider $|\hat{X}|_K$ in Eq. (39), then the probability of deviation from the true set size, at a given distance $t \geq 0$, is

$$
P\left(|\hat{X}| - |X| \leq t\right) = I_{u(|X|, k, t/|X|)}(k, |X| - k + 1) - I_{l(|X|, k, t/|X|)}(k, |X| - k + 1)
$$

where $u(|X|, k, t/|X|) = k - 1 - |X| - t$ and $l(|X|, k, t/|X|) = k - 1 - |X| - t$

and $I_x(a, b)$ is the regularized incomplete beta function.

In the case of a KMV estimator bound, we can evaluate:

$$
I_x(k, |X| - k + 1) = \sum_{i=k}^{|X|} \binom{|X|}{i} x^i(1-x)^{|X|-i}
$$

2) Set Intersection $|X \cap Y|$: Given $K_X$ and $K_Y$ of size $k_X$ and $k_Y$, one can construct a KMV $K_{X \cup Y}$ by taking the $k = \min\{k_X, k_Y\}$ smallest elements from $K_X \cup K_Y$. $|\hat{X} \cap \hat{Y}|_K$ and $|\hat{Y}|_K$ can be computed using the following equations (note that the second one uses the exact sizes of $X, Y$ instead of their estimators).

$$
|\hat{X} \cap \hat{Y}|_K = |\hat{X}|_K + |\hat{Y}|_K - |\hat{X} \cup \hat{Y}|_K
$$

(40)

$$
|\hat{X} \cap \hat{Y}|_K = |X| + |Y| - |\hat{X} \cup \hat{Y}|_K
$$

(41)

We present now a simple upper bound (using the union bound) on the probability that $|\hat{X} \cap \hat{Y}|_K$ deviates by more than $t$ from the true value.

**Proposition A.8.** Let $|\hat{X} \cap \hat{Y}|_K$ be the estimator defined in (40), then the following upper bound for the probability of deviation from the true intersection set size, at a given distance $t \geq 0$, holds:

$$
P\left(||\hat{X} \cap \hat{Y}|_K - |X \cap Y|| \geq t\right) \leq P(||\hat{X}|_K - |X|| \geq t/3) + P(||\hat{Y}|_K - |Y|| \geq t/3) + P(||\hat{X} \cup \hat{Y}|_K - |X \cup Y|| \geq t/3)
$$

where the probabilities on the right can be evaluated with Proposition A.7.

Yet, if we know the exact size of $X$ and $Y$ (a reasonable assumption for graph algorithms as the degrees can be easily precomputed), we can get a considerably better bound.

**Proposition A.9.** Let $|\hat{X} \cap \hat{Y}|_K$ be the estimator from (41), then

$$
P\left(||\hat{X} \cap \hat{Y}|_K - |X \cap Y|| \geq t\right) =
$$

$$
I_{u(|X \cap Y|, k, t/|X \cap Y|)}(k, |X \cup Y| - k + 1) - I_{l(|X \cap Y|, k, t/|X \cap Y|)}(k, |X \cup Y| - k + 1)
$$

The bound presented above is a simple application of the identity $|X \cap Y| = |X| + |Y| - |X \cup Y|$ and Proposition A.7 on the estimator of $|X \cup Y|$. 

9uniformly at random