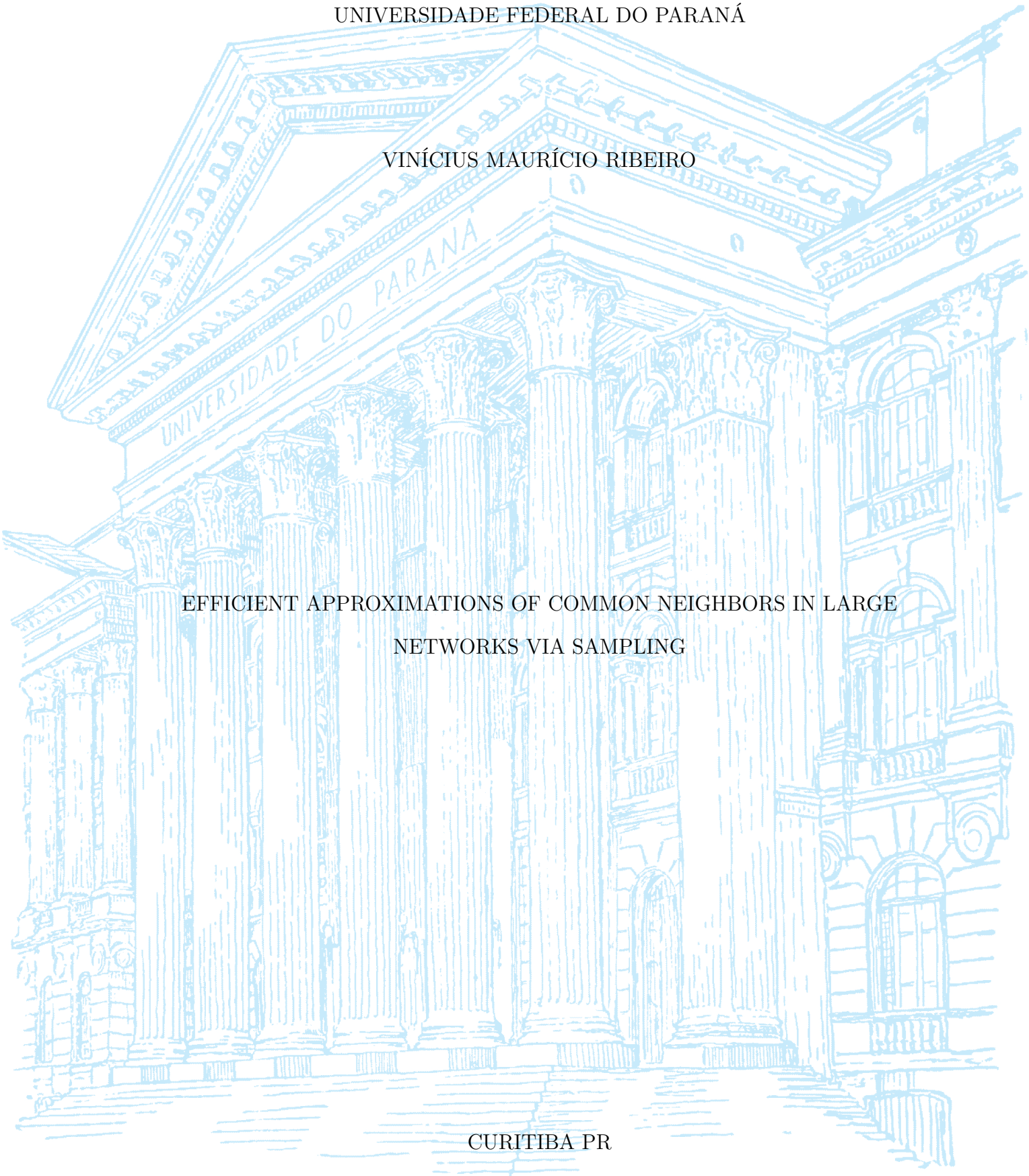
UNIVERSIDADE FEDERAL DO PARANÁ

VINÍCIUS MAURÍCIO RIBEIRO



EFFICIENT APPROXIMATIONS OF COMMON NEIGHBORS IN LARGE NETWORKS VIA SAMPLING

CURITIBA PR

2025

VINÍCIUS MAURÍCIO RIBEIRO

EFFICIENT APPROXIMATIONS OF COMMON NEIGHBORS IN LARGE NETWORKS VIA SAMPLING

Trabalho apresentado como requisito parcial à conclusão do Curso de Bacharelado em Ciência da Computação, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Computação.*

Orientador: André Luís Vignatti.

CURITIBA PR

2025

RESUMO

Vizinhos Comuns (CN) é uma quantidade fundamental na análise de redes e mineração de dados, servindo de base para várias métricas e medidas de similaridade. Neste trabalho, abordamos o problema de contar aproximadamente o número de vizinhos comuns para todos os pares de vértices em um grafo. Como o cálculo exato aumenta com o tamanho do grafo, ele se torna computacionalmente proibitivo para redes de larga escala. Para superar essa limitação, propomos algoritmos eficientes baseados em amostragem que aproximam CN com fortes garantias probabilísticas. Com base nos resultados da teoria da dimensão VC, nossos algoritmos obtêm tamanhos de amostra que são independentes do tamanho do grafo e dependem apenas de seu grau máximo, oferecendo uma solução prática e escalável para análise de redes de larga escala. Introduzimos seis algoritmos, categorizados por normalização (normalizado ou não normalizado) e método de amostragem (baseado em vértice, aresta e *wedge*). Três algoritmos calculam CN normalizado com garantias de erro aditivo, enquanto os outros lidam com CN não normalizado com garantias de erro multiplicativo. Resultados experimentais demonstram que nossos algoritmos alcançam acelerações substanciais em relação aos métodos exatos, mantendo alta precisão.

Palavras-chave: Análise de redes sociais. Vizinhos Comuns. Algoritmo aleatorizado.

ABSTRACT

Common Neighbors (CN) is a fundamental quantity in network analysis and data mining, underpinning various metrics and similarity measures. In this paper, we address the problem of approximately counting the number of common neighbors for all vertex pairs in a graph. Since exact computation scales with the size of the graph, it becomes computationally prohibitive for large-scale networks. To overcome this limitation, we propose efficient sampling-based algorithms that approximate CN with strong probabilistic guarantees. Building on results from VC-dimension theory, our algorithms obtain sample sizes that are independent of the size of the graph and depend only on its maximum degree, offering a practical and scalable solution for large-scale network analysis. We introduce six algorithms, categorized by normalization (normalized or unnormalized) and sampling method (vertex-, edge- and wedge-based). Three algorithms compute normalized CN with additive error guarantees, while the others handle unnormalized CN with multiplicative error guarantees. Experimental results demonstrate that our algorithms achieve substantial speedups over exact methods while maintaining high accuracy.

Keywords: Social network analysis. Common Neighbors. Randomized algorithm.

CONTENTS

1 INTRODUCTION

The *common neighbors* (CN) of two vertices in a graph is the size of the intersection of their neighborhoods i.e., $|N(u) \cap N(v)|$, where $N(u)$ and $N(v)$ are the neighborhoods of vertices u and v, respectively. It plays a central role in determining several metrics and similarity measures used in network analysis, as discussed in reference textbooks such as (Easley and Kleinberg, 2010; Newman, 2010; Menczer et al., 2020) (see Section 2 for key definitions and results used throughout the paper).

Kleinberg and Easley (Easley and Kleinberg, 2010) propose several metrics that use the concept of common neighbors. The neighborhood overlap metric (Easley and Kleinberg, 2010, Section 3.3) is defined as the ratio of common neighbors to the neighborhood union, while the embeddedness metric (Easley and Kleinberg, 2010, Section 3.5) is simply the number of common neighbors of two adjacent vertices. In another context (Easley and Kleinberg, 2010, Section 4.4), the neighborhood overlap is used to assess similarity within bipartite affiliation networks. Menczer, Fortunato, and Davis (Menczer et al., 2020, Section 6.2) coined the term "structural equivalence", which essentially duplicates the definition of the neighborhood overlap metric from Kleinberg and Easley. Newman (Newman, 2010) introduces an equation that calculates the number of paths of length two between vertices u and v (Newman, 2010, Section 6.10), and this value is precisely the number of common neighbors. Additionally, in the study of similarity, Newman introduces the concept of structural equivalence (Newman, 2010, Section 7.12), which also represents CN. A widely used metric is cosine similarity (Newman, 2010, Section 7.2.1), which, in the network context, is computed by dividing the number of common neighbors of two vertices by the geometric mean of their degrees. Another measure of structural equivalence is the so-called Euclidean distance (not to be confused with the more common use of this term), which uses CN in its calculation (Newman, 2010, Section 7.12.3).

1.1 OUR RESULTS

We present sampling-based algorithms that use notions from VC-dimension theory to derive sample sizes independent of the graph's size, depending instead on its maximum degree. Six algorithms are presented, categorized by normalization (normalized or unnormalized) and sampling method (vertex-, edge- or wedge-based).

The first three algorithms compute three normalized variants of CN: *vertex-normalized*, *edge-normalized* and *wedge-normalized* common neighbors. Normalization is useful since it provides values relative to the graph, which can be more informative than absolute measures. Specifically, vertex normalization highlights the proportion of vertices involved in the neighborhood (e.g., "50% of the vertices of the graph lie in the common neighborhood of u and v"), while edge and wedge normalizations emphasize the participation of edges and wedges, respectively. These algorithms guarantee that, for all vertex pairs, the estimated values are within an *additive error* ϵ of the true values with probability at least $1 - \delta$.

Next, we present three algorithms for the unnormalized case. These algorithms are constructed by denormalizing the vertex-, edge- and wedge-normalized algorithms described earlier. These unnormalized algorithms guarantee a *multiplicative error* ϵ with

probability at least $1-\delta$, provided the normalized CN exceeds a threshold η. This threshold provides robustness for small CNs, where multiplicative error bounds become impractical.

Both vertex-, edge- and wedge-based strategies are presented because neither consistently outperforms the other; instead, there is a trade-off between estimator quality and running time. On the one hand, vertex sampling has the higher running time of $\mathcal{O}(\Delta^2 \lg \Delta + |E|)$, followed by edge sampling with $\mathcal{O}(\Delta \lg \Delta + |E|)$ and wedge sampling with running time of $O(|E|)$, where Δ is the maximum degree of the graph. On the other hand, we show that the vertex-based sampling estimator is generally more efficient, followed by the edge- and wedge-sampling strategies, respectively.

Our experiments are divided into three parts: an evaluation of the normalized versions, an analysis of the denormalized versions, and an empirical study of the parameter η. Results show that all algorithms achieve significantly faster running times than exact methods, while meeting theoretical quality guarantees.

To the best of our knowledge, these are the first algorithms to compute CN for all pairs with sample sizes independent of the graph size.

1.2 RELATED WORK

The current state-of-the-art implementation for computing CN for all pairs is by An et al. (An et al., 2019). This algorithm runs in $O(\Delta|E|)$ and serves as a basis for comparison for our probabilistic algorithms. Its strategy primarily focuses on counting all the wedges in the graph (further details can be found in Section 5). Other strategies for computing CN can also be found in the literature. Besta et al. (Besta et al., 2021) suggest that many graph mining algorithms can be largely expressed using simple set operations, such as intersection and union, on sets of vertices or edges. In a subsequent 2022 work, Besta et al. (Besta et al., 2022) note that graphs can be effectively represented as collections of vertex and edge sets. To efficiently encode these sets, they use probabilistic set representations such as Bloom filters (Bloom, 1970). They further show that many graph algorithm operations can be expressed using set intersection cardinality, $|X \cap Y|$, and design estimators $|\widehat{X \cap Y}|$ that approximate $|X \cap Y|$. Similar to the estimators proposed in our work, their estimator also incorporates error and confidence parameters, suggesting a potential generalization of our work. However, their work focuses on efficiently computing a single intersection, while our results address multiple intersections. Our approach is more efficient, for example, when computing a number of intersections proportional to or exceeding the maximum degree.

Sample complexity techniques based on the Vapnik-Chervonenkis (VC) dimension (Vapnik and Chervonenkis, 1971; Shalev-Shwartz and Ben-David, 2014), originating from computational learning theory, have recently found applications in graph algorithms (Riondato and Kornaropoulos, 2014, 2016; de Lima et al., 2020, 2022). Notably, Riondato and Kornaropoulos (Riondato and Kornaropoulos, 2014, 2016) pioneered this approach, developing a fast algorithm for computing betweenness centrality. Building on this work, Lima et al. (de Lima et al., 2020) addressed percolation centrality, a generalization of betweenness centrality. They further applied these techniques to the computation of clustering coefficients, a key problem in graph mining (de Lima et al., 2022). Combining VC dimension with Rademacher averaging and progressive sampling offers further potential improvements, as demonstrated in several studies of previously addressed problems (Riondato and Upfal, 2018; Pellegrina and Vandin, 2023; Cousins et al., 2023; de Lima et al., 2022).

2 PRELIMINARIES

This section presents key definitions and results used throughout the paper.

2.1 GRAPH THEORY

Let $G = (V, E)$ be an undirected graph. For each $v \in V$, let δ_v be the degree of vertex v, and let $\Delta = \max_{v \in V} \delta_v$ be the maximum degree of G. The *neighborhood* $N(v)$ of a vertex v is the set of all vertices adjacent to v in G. A *wedge* is a path of length two, i.e., a sequence of three vertices $u, v, t \in V$ such that $\{u, v\}, \{v, t\} \in E$. We do not differentiate the wedges by the order in which we traverse the path, i.e., $(u, v, t) = (t, v, u)$. W is the set of all wedges in G.

Definition 1. *The* common neighbors (CN) *metric $c(u, v)$ of two vertices u and v is the size of the intersection of their neighborhoods, i.e., $c(u, v) = |N(u) \cap N(v)|$.*

Definition 2. *The* vertex-normalized CN *$c_{vertex}(u, v)$ of two vertices u and v is given by $c_{vertex}(u, v) = \frac{c(u,v)}{|V|}$.*

Definition 3. *The* edge-normalized CN *$c_{edge}(u, v)$ of two vertices u and v is given by $c_{edge}(u, v) = \frac{2c(u,v)}{|E|}$.*

Definition 4. *The* wedge-normalized CN *$c_{wedge}(u, v)$ of two vertices u and v is given by $c_{wedge}(u, v) = \frac{c(u,v)}{|W|}$.*

In Definition 3, the multiplicative factor of two accounts for the two edges (one from u and one from v) associated with each common neighbor in undirected graphs. This factor, while not strictly required, simplifies subsequent calculations.

The following lemmas provide a way to count the number of wedges in a graph.

Proposition 1. *Let $G = (V, E)$ be a graph. The number of wedges in G is given by*

$$|W| = \sum_{\{u,v\} \in E} \frac{\delta_u + \delta_v - 2}{2}.$$

Proof. An edge $\{u, v\} \in E$ is part of a wedge if u is the center (e.g., (x, u, v)) or v is the center (e.g., (u, v, y)). If u is the center, there are $\delta_u - 1$ choices for x (excluding v). If v is the center, there are $\delta_v - 1$ choices for y (excluding u). Therefore, the number of wedges in G that contain $\{u, v\}$ is $(\delta_u - 1) + (\delta_v - 1) = \delta_u + \delta_v - 2$. When summing the contribution of all edges, each wedge is counted twice (once for each edge in it), so we divide the total by 2. □

Proposition 2. *Let $G = (V, E)$ be a graph. The number of wedges in G is given by*

$$|W| = \sum_{v \in V} \binom{\delta_v}{2}.$$

Proof. Consider a vertex $v \in V$. By choosing two neighbors u and w from $N(v)$, we can form a wedge (u, v, w). The number of ways to choose two vertices from $N(v)$ is given by $\binom{\delta_v}{2}$, so there are $\binom{\delta_v}{2}$ wedges centered at v. Summing this over all vertices in V gives us the total number of wedges in the graph. □

2.2 VC DIMENSION

Let Ω be a sample space, and $\mathcal{E} = \{E_1, E_2, ..., E_n\}$ a set of events (subsets) over Ω. Consider the task of approximating the probability of each event $p_{E_i} = \Pr(E_i)$ by sampling from Ω. Let $S = (s_1, s_2, \ldots, s_m) \in \Omega^m$ be a sample of size m. We estimate p_{E_i} using the *relative frequency* of E_i, i.e.,

$$\hat{p}_{E_i} = \frac{1}{m} \sum_{j=1}^{m} \mathbb{1}_{E_i}(s_j),$$

where $\mathbb{1}_{E_i}(s_j)$ is an indicator function that is equal to 1 if s_j occurs in E_i (i.e., $s_j \in E_i$), and 0 otherwise. We are interested in determining the sample size m such that

$$\Pr\left(\forall i,\ |\hat{p}_{E_i} - p_{E_i}| \leq \epsilon\right) \geq 1 - \delta, \tag{2.1}$$

where ϵ is the margin of error and $1 - \delta$ is the confidence level.

Suppose we have individual bounds $\Pr(|\hat{p}_{E_i} - p_{E_i}| \leq \epsilon)$ for each event E_i, obtained using well-known inequalities like Chernoff or Hoeffding bounds. We can combine these individual bounds to obtain our desired global bound using the union bound. The downside with this approach is that the union bound results in a sample size dependent on $|\mathcal{E}|$. The Vapnik-Chervonenkis (VC) dimension offers a more refined method, bounding the sample size based on the "complexity" of the set of events $\mathcal{E}$, rather than its cardinality.

A *range space* $(X, \mathcal{R})$ consists of a set X and a family $\mathcal{R}$ of subsets of X. The *projection* of $A \subseteq X$ on $\mathcal{R}$ is $P_{\mathcal{R}}(A) = \{A \cap R : R \in \mathcal{R}\}$. A set A is *shattered* by $\mathcal{R}$ if $P_{\mathcal{R}}(A) = 2^A$.

Definition 5. *The* VC dimension *of a range space $(X, \mathcal{R})$, denoted $d_{VC}(\mathcal{R})$, is the size of the largest set $A \subseteq X$ shattered by $\mathcal{R}$.*

For an in-depth explanation of VC dimension, see Shalev-Schwartz and Ben-David (Shalev-Shwartz and Ben-David, 2014), and Mitzenmacher and Upfal (Mitzenmacher and Upfal, 2017).

2.3 SAMPLE COMPLEXITY

We now present the definitions and theorems that will be used to bound the sample size necessary to obtain confidence intervals for the probability of each event of interest using the VC dimension. It is possible to derive bounds that ensure approximations with absolute or relative error.

Definition 6. *Let $(X, \mathcal{R})$ be a range space with a probability distribution π over X, $p_R = \Pr_\pi(R)$ the probability of a range $R \in \mathcal{R}$, and $\hat{p_R}$ the relative frequency of R based on a sample S. For $\epsilon \in (0, 1)$, S is an* (absolute) ϵ-approximation *to $(X, \mathcal{R})$ if*

$$|\hat{p_R} - p_R| \leq \epsilon, \quad \forall R \in \mathcal{R}.$$

Theorem 1 (see (Har-Peled and Sharir, 2011), Thm. 2.12)**.** *Let $(X, \mathcal{R})$ be a range space with VC dimension $d_{VC}(\mathcal{R}) \leq d$, and let π be a probability distribution on X. For $\epsilon, \delta \in (0, 1)$, and a sample S of size m drawn from π, with probability at least $1 - \delta$, S is an ϵ-approximation to $(X, \mathcal{R})$ if*

$$m \geq \frac{b}{\epsilon^2}\left(d + \ln\frac{1}{\delta}\right),$$

where b is a positive constant.

Definition 7 (see (Har-Peled and Sharir, 2011))**.** *Let $(X, \mathcal{R})$ be a range space with a probability distribution π over X, $p_R = \Pr_\pi(R)$ the probability of a range $R \in \mathcal{R}$, and $\hat{p_R}$ the relative frequency of R based on a sample S. For $\epsilon \in (0, 1)$, S is a* relative (η, ϵ)-approximation *to $(X, \mathcal{R})$ if, for all $R \in \mathcal{R}$,*

$$|\hat{p_R} - p_R| \leq \epsilon \max\{p_R, \eta\}.$$

Introducing the new parameter η for the relative approximation might seem unintuitive compared to the absolute ϵ-approximation. As explained by Har-Peled and Sharir (Har-Peled and Sharir, 2011), as p_R approaches zero, the precision of the relative approximation must increase. When $p_R = 0$, the approximation would need to be exact, which is impossible to guarantee, unless $S = X$. The parameter η addresses this issue by explicitly setting the approximation precision for events with small probability. We therefore refer to η as the *threshold parameter*.

Theorem 2 (see (Li et al., 2001), Thm. 5)**.** *Let $(X, \mathcal{R})$ be a range space with VC dimension $d_{VC}(\mathcal{R}) \leq d$, and let π be a probability distribution on X. For $\eta, \epsilon, \delta \in (0, 1)$, and a sample S of size m drawn from π, with probability at least $1-\delta$, S is a relative (η, ϵ)-approximation to $(X, \mathcal{R})$ if*

$$m \geq \frac{b'}{\epsilon^2 \eta}\left(d \ln \frac{1}{\eta} + \ln \frac{1}{\delta}\right),$$

where b' is a positive constant.

The values used for the constants b and b' are discussed in Section 5.

2.4 SHATTER POTENTIAL

Suppose we have two range spaces, $(X, \mathcal{R}_1)$ and $(X, \mathcal{R}_2)$, where $d_{\mathrm{VC}}(\mathcal{R}_1) \leq M$ and $d_{\mathrm{VC}}(\mathcal{R}_2) \leq N$, with $M > N$. Although we cannot conclude that $\mathcal{R}_1$ is inherently more complex than $\mathcal{R}_2$ (i.e., $d_{\mathrm{VC}}(\mathcal{R}_1) > d_{\mathrm{VC}}(\mathcal{R}_2)$ does not hold by default), $\mathcal{R}_1$ possesses a higher *potential* for complexity than $\mathcal{R}_2$. To formalize this notion of potential complexity, we introduce the concept of the *shatter potential* of a range space $(X, \mathcal{R})$.

Definition 8. *The* shatter potential of an element $x \in X$ *with respect to a range space $(X, \mathcal{R})$ is the number of sets in $\mathcal{R}$ that contain x, i.e., $\sigma_{\mathcal{R}}(x) = |\{R \in \mathcal{R} \mid x \in R\}|$.*

Definition 9. *The* shatter potential of a range space $(X, \mathcal{R})$ *is the maximum shatter potential of any element in X, i.e., $\sigma(\mathcal{R}) = \max_{x \in X} \sigma_{\mathcal{R}}(x)$.*

While it may not be immediately clear whether Definition 9 truly captures the notion of potential complexity, Theorem 3 provides a theoretical justification for this definition.

Theorem 3. *Let $(X, \mathcal{R})$ be a range space. Then, $d_{VC}(\mathcal{R}) \leq \lfloor \lg(\sigma(\mathcal{R})) \rfloor + 1$.*

Proof. Let A be the largest set shattered by $\mathcal{R}$, so $d_{\mathrm{VC}}(\mathcal{R}) = |A|$. The projection $P_{\mathcal{R}}(A)$ is equal to the power set of A, and thus $|P_{\mathcal{R}}(A)| = 2^{|A|} = 2^{d_{\mathrm{VC}}(\mathcal{R})}$. Fix an arbitrary element $a \in A$. Each subset of A either contains a or does not, so a appears in exactly half of the subsets in 2^A. This implies a appears in $2^{|A|-1} = 2^{d_{\mathrm{VC}}(\mathcal{R})-1}$ subsets. Thus, $\sigma_{\mathcal{R}}(a) = 2^{d_{\mathrm{VC}}(\mathcal{R})-1}$, which implies $\sigma(\mathcal{R}) \geq 2^{d_{\mathrm{VC}}(\mathcal{R})-1}$. The result follows by isolating $d_{\mathrm{VC}}(\mathcal{R})$. □

Given $\sigma(\mathcal{R}_1) = M > \sigma(\mathcal{R}_2) = N$, we have that $d_{\mathrm{VC}}(\mathcal{R}_1) \leq \lfloor \lg M \rfloor + 1$ and $d_{\mathrm{VC}}(\mathcal{R}_2) \leq \lfloor \lg N \rfloor + 1$. Since $M > N$, it follows that $\lfloor \lg M \rfloor + 1 > \lfloor \lg N \rfloor + 1$, implying that $\mathcal{R}_1$ exhibits a higher potential complexity than $\mathcal{R}_2$.

It is important to note that the concept of shatter potential is not new; it has been used implicitly in previous research (de Lima et al., 2022, Proof of Theorem 2), (de Lima et al., 2022, Proof of Theorem 5), and (Riondato and Kornaropoulos, 2014, Proof of Lemma 1). We simply formalized it and gave it a name to allow for more effective use in our analysis.

The shatter potential is a particularly useful tool for quickly determining an upper bound for the VC dimension. In conjunction with Theorems 1 and 2, we can determine a sample size that depends solely on the complexity of the event set, rather than its size.

3 NORMALIZED ESTIMATIVES

Our primary approach to estimate CN is to denormalize either vertex-, edge- or wedge-normalized CN. To achieve this, we first present algorithms for the normalized CN in Sections 3.1, 3.2 and 3.3, respectively. In Section 4, we then demonstrate how to estimate $c(u, v)$ from $c_{\text{vertex}}(u, v)$, $c_{\text{edge}}(u, v)$ and $c_{\text{wedge}}(u, v)$.

3.1 ESTIMATION VIA VERTEX NORMALIZATION

In this section, we present our strategy for estimating the vertex-normalized CN $c_{\text{vertex}}(u, v)$ for all $u, v \in V$.

3.1.1 Range Space and VC Dimension Results

Let the set of vertices V be our sample space, and $E_{\text{vertex}}(u, v)$ the event of sampling a vertex in $N(u) \cap N(v)$, that is, $E_{\text{vertex}}(u, v) = N(u) \cap N(v)$. Then,

$$\Pr\Big(E_{\text{vertex}}(u, v)\Big) = \frac{|N(u) \cap N(v)|}{|V|} = c_{\text{vertex}}(u, v),$$

where the last equation follows from Definition 2. Thus, by sampling vertices from the graph, we approximate $c_{\text{vertex}}(u, v)$ by estimating the probability of $E_{\text{vertex}}(u, v)$.

Let $(V, \mathcal{R}_{\text{vertex}})$ be a range space, where $\mathcal{R}_{\text{vertex}}$ is the set of all events $E_{\text{vertex}}(u, v)$ for each pair of vertices $u, v \in V$, i.e.,

$$\mathcal{R}_{\text{vertex}} = \Big\{E_{\text{vertex}}(u, v) \mid u, v \in V\Big\}.$$

Theorem 4 bounds the VC dimension of $(V, \mathcal{R}_{\text{vertex}})$.

Theorem 4. *The range space $(V, \mathcal{R}_{vertex})$ has $d_{VC}(\mathcal{R}_{vertex}) \leq \lfloor 2 \lg(\Delta) \rfloor$.*

Proof. To derive the bound for the VC dimension, we examine the shatter potential of $(V, \mathcal{R}_{\text{vertex}})$. Let $v \in V$ be an arbitrary vertex, and $N(v) = \{n_1, n_2, \ldots, n_{\delta_v}\}$ the neighbors of v. Since $v \in N(n_i) \cap N(n_j)$ for all pairs $n_i, n_j \in N(v)$, vertex v is included in $\binom{\delta_v}{2}$ events $E_{\text{vertex}}(n_i, n_j)$, yielding $\sigma_{\mathcal{R}_{\text{vertex}}}(v) = \binom{\delta_v}{2}$. The shatter potential of $(V, \mathcal{R}_{\text{vertex}})$ is thus

$$\sigma\Big(\mathcal{R}_{\text{vertex}}\Big) = \max_{v \in V} \binom{\delta_v}{2} = \binom{\Delta}{2}.$$

By Theorem 3,

$$d_{\text{VC}}(\mathcal{R}_{\text{vertex}}) \leq \left\lfloor \lg \binom{\Delta}{2} \right\rfloor + 1 \leq \left\lfloor \lg \left(\frac{\Delta^2}{2} \right) \right\rfloor + 1 = \lfloor 2 \lg(\Delta) \rfloor.$$

□

3.1.2 Algorithm

The algorithm involves sampling vertices from the graph to estimate $c_{\text{vertex}}(u, v)$ for each pair $u, v \in V$. Let $S = (s_1, s_2, \ldots, s_m) \in \Omega^m$ be a i.i.d. sample of size m, where each s_i is a vertex in V. We use the relative frequency of $E_{\text{vertex}}(u, v)$ as an estimator for $c_{\text{vertex}}(u, v)$, i.e.,

$$\hat{c}_{\text{vertex}}(u, v) = \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}_{E_{\text{vertex}}(u,v)}(s_i), \tag{3.1}$$

where $\mathbb{1}_{E_{\text{vertex}}(u,v)}(s_i)$ is an indicator function that is equal to 1 if vertex $s_i \in N(u) \cap N(v)$, and 0 otherwise. Note that $\mathbb{E}\left[\hat{c}_{\text{vertex}}(u, v)\right] = c_{\text{vertex}}(u, v)$, i.e., $\hat{c}_{\text{vertex}}(u, v)$ is an unbiased estimator for $c_{\text{vertex}}(u, v)$.

The algorithm takes as input a graph $G = (V, E)$, and the accuracy and confidence parameters $0 < \epsilon, \delta < 1$, assumed to be constants. It outputs the estimate $\hat{c}_{\text{vertex}}(u, v)$ for all pairs $u, v \in V$. Algorithm 1 presents the pseudo-code of the vertex-normalized CN algorithm. The value for the constant b used in the evaluation of the sample size is discussed further in Section 5.

Algorithm 1: VertexNormalizedCN(G, ε, δ)

input : Graph $G = (V, E)$, accuracy ϵ, confidence $1 - \delta$
output : Estimates $\hat{c}_{\text{vertex}}(u, v)$ for all pairs $u, v \in V$
Initialize all $\hat{c}_{\text{vertex}}[u, v]$ to zero
$\Delta \leftarrow$ getMaximumDegree(G)
$m \leftarrow \left\lceil \frac{b}{\epsilon^2} \left(\lfloor 2 \lg(\Delta) \rfloor + \ln \frac{1}{\delta} \right) \right\rceil$
for $i \leftarrow 1$ **to** m **do**
    $s \leftarrow$ sampleVertex(G)
    forall *pairs* $n_i, n_j \in N(s)$ **do**
        $\hat{c}_{\text{vertex}}[n_i, n_j] \leftarrow \hat{c}_{\text{vertex}}[n_i, n_j] + \frac{1}{m}$
return $\hat{c}_{\text{vertex}}[n_i, n_j]$ for all pairs $u, v \in V$

Theorem 5. *Given a graph $G = (V, E)$, accuracy ϵ, and confidence $1 - \delta$, the estimates $\hat{c}_{vertex}(u, v)$ obtained by Algorithm 1 satisfy*

$$\Pr\left(\forall u, v \in V,\ |c_{vertex}(u, v) - \hat{c}_{vertex}(u, v)| \leq \epsilon \right) \geq 1 - \delta.$$

Proof. Note that the algorithm returns estimations that replicate the behavior of Equation 3.1. By Theorem 4, the VC dimension is at most $\lfloor 2 \lg(\Delta) \rfloor$. Together with Theorem 1, the sample with size

$$m = \left\lceil \frac{b}{\epsilon^2} \left(\lfloor 2 \lg(\Delta) \rfloor + \ln \frac{1}{\delta} \right) \right\rceil,$$

is an ϵ-approximation to $(V, \mathcal{R}_{\text{vertex}})$ with confidence $1 - \delta$. Thus, for all $u, v \in V$, we have $|\hat{c}_{\text{vertex}}(u, v) - c_{\text{vertex}}(u, v)| \leq \epsilon$, with probability at least $1 - \delta$. □

Theorem 6. *Algorithm 1 has running time $\mathcal{O}(\Delta^2 \lg \Delta + |E|)$.*

Proof. To avoid explicitly iterating through all (u, v) pairs and setting them to zero, we can use a direct addressing table indexed by (u, v). This table is used during execution as

follows: if the value at index (u, v) is null, it is set to zero; otherwise, it is incremented according to the algorithm.

Computing Δ in the preprocessing phase requires $\mathcal{O}(|E|)$ time. For each sampled vertex, $\mathcal{O}(\Delta^2)$ estimates are updated. As we sample m vertices in total, the time complexity associated with these updates is $\mathcal{O}(m \cdot \Delta^2)$ and since $m = \mathcal{O}(\log \Delta)$, the result follows. □

3.2 ESTIMATION VIA EDGE NORMALIZATION

In this section, we present our strategy for estimating the edge-normalized CN $c_{\text{edge}}(u, v)$ for all $u, v \in V$.

3.2.1 Range Space and VC Dimension Results

Let the set of edges E be our sample space. For each pair of vertices $u, v \in V$, we define an event $E_{\text{edge}}(u, v)$. This event is a set containing the edges $\{t, u\}$ and $\{t, v\}$ for all vertices t that are common neighbors of both u and v. Formally,

$$E_{\text{edge}}(u, v) = \Big\{\{u, t\}, \{v, t\} \mid t \in N(u) \cap N(v)\Big\}.$$

Figure 3.1 presents an example of the events.

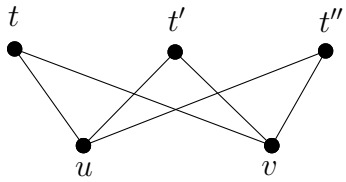


Figure 3.1: An example of the event $E_{\text{edge}}(u, v)$. Here, u and v have three neighbors in common, t, t' and t''. Therefore, the event is composed of the six edges depicted in the figure.

The probability of sampling an edge in $E_{\text{edge}}(u, v)$ is then

$$\begin{aligned}
\Pr\left(E_{\text{edge}}(u, v)\right) &= \Pr\left(\Big\{\{u, t\}, \{v, t\} \mid t \in N(u) \cap N(v)\Big\}\right) \\
&= \Pr\left(\Big\{\{u, t\} \mid t \in N(u) \cap N(v)\Big\}\right) \\
&\quad + \Pr\left(\Big\{\{v, t\} \mid t \in N(u) \cap N(v)\Big\}\right) \\
&= \sum_{t \in N(u) \cap N(v)} \Pr(\{u, t\}) + \sum_{t \in N(v) \cap N(u)} \Pr(\{v, t\}) \\
&= \frac{|N(u) \cap N(v)|}{|E|} + \frac{|N(v) \cap N(u)|}{|E|} = c_{\text{edge}}(u, v),
\end{aligned}$$

where the first equation follows from the definition of the event, the second and third from the fact that the events are disjoint, and the last from Definition 3. Thus, by sampling edges, we approximate $c_{\text{edge}}(u, v)$ by estimating the probability of $E_{\text{edge}}(u, v)$.

Let $(E, \mathcal{R}_{\text{edge}})$ be a range space, where $\mathcal{R}_{\text{edge}}$ is the set of all events $E_{\text{edge}}(u, v)$ for each pair of vertices $u, v \in V$, i.e.,

$$\mathcal{R}_{\text{edge}} = \Big\{E_{\text{edge}}(u, v) \mid u, v \in V\Big\}.$$

Theorem 7 bounds the VC dimension of $(E, \mathcal{R}_{\text{edge}})$.

Theorem 7. *The range space $(E, \mathcal{R}_{edge})$ has $d_{VC}(\mathcal{R}_{edge}) \leq \lfloor \lg \Delta \rfloor + 2$.*

Proof. Consider an arbitrary edge $e = \{u, v\} \in E$. Let $N'(u) = \{u_1, u_2, \ldots, u_{\delta_u}\} \setminus \{v\}$ and $N'(v) = \{v_1, v_2, \ldots, v_{\delta_v}\} \setminus \{u\}$ denote the set of neighbors of u and v respectively, excluding the opposite vertex. Observe that $e \in E_{\text{edge}}(u_i, v) \cup E_{\text{edge}}(v_i, u)$ for all $u_i \in N'(u)$ and $v_i \in N'(v)$. Consequently, e belongs to $(\delta_u - 1) + (\delta_v - 1)$ events, yielding $\sigma_{\mathcal{R}_{\text{edge}}}(e) = (\delta_u - 1) + (\delta_v - 1)$. Thus, the shatter potential of $\left(E, \mathcal{R}_{\text{edge}}\right)$ is

$$\sigma\left(\mathcal{R}_{\text{edge}}\right) = \max_{\{u,v\} \in E} (\delta_u + \delta_v - 2) \leq 2\Delta - 2.$$

By Theorem 3,

$$d_{\text{VC}}\left(\mathcal{R}_{\text{edge}}\right) \leq \lfloor \lg(2\Delta - 2) \rfloor + 1 \leq \lfloor \lg \Delta \rfloor + 2.$$

□

3.2.2 Algorithm

Given a collection of i.i.d. samples (edges) $S = (s_1, s_2, \ldots, s_m) \in E^m$, we use the relative frequency of $E_{\text{edge}}(u, v)$ as an estimator for $\Pr(E_{\text{edge}}(u, v)) = c_{\text{edge}}(u, v)$, i.e.,

$$\hat{c}_{\text{edge}}(u, v) = \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}_{E_{\text{edge}}(u,v)}(s_i) \tag{3.2}$$

where $\mathbb{1}_{E_{\text{edge}}(u,v)}(s_i)$ is an indicator function that is equal to 1 if edge $s_i \in E_{\text{edge}}(u, v)$, and 0 otherwise. Also, note that $\hat{c}_{\text{edge}}(u, v)$ is an unbiased estimator.

The input of the algorithm takes is a graph $G = (V, E)$ and parameters $0 < \epsilon, \delta < 1$. It outputs the estimate $\hat{c}_{\text{edge}}(u, v)$ for all pairs $u, v \in V$. For each sampled edge $\{u, v\}$, we update all estimates $\hat{c}_{\text{edge}}(u_i, v)$ and $\hat{c}_{\text{edge}}(v_i, u)$ for each neighbor $u' \in N(u) \setminus \{v\}$ and $v' \in N(v) \setminus \{u\}$. Consequently, the computational complexity per sample is $\mathcal{O}(\delta_u + \delta_v) = \mathcal{O}(\Delta)$, which, notably, is asymptotically lower than that required for the vertex-normalized CN. Algorithm 2 presents the pseudo-code of the edge-normalized CN algorithm. The value for the constant b used in the evaluation of the sample size is discussed further in Section 5.

Algorithm 2: EdgeNormalizedCN(G, ε, δ)

input : Graph $G = (V, E)$, accuracy ϵ, confidence $1 - \delta$
output : Estimates $\hat{c}_{\text{edge}}(u, v)$ for all pairs of vertices $u, v \in V$
Initialize all $\hat{c}_{\text{edge}}[u, v]$ to zero
$\Delta \leftarrow$ getMaximumDegree(G)
$m \leftarrow \left\lceil \frac{b}{\epsilon^2} \left(\lfloor \lg \Delta \rfloor + 2 + \ln\left(\frac{1}{\delta}\right) \right) \right\rceil$
for $i \leftarrow 1$ **to** m **do**
  $\{u, v\} \leftarrow$ sampleEdge(G)
  for *each* $u' \in N(u) \setminus \{v\}$ **do**
    $\hat{c}_{\text{edge}}[u', v] \leftarrow \hat{c}_{\text{edge}}[u', v] + \frac{1}{m}$
  for *each* $v' \in N(v) \setminus \{u\}$ **do**
    $\hat{c}_{\text{edge}}[v', u] \leftarrow \hat{c}_{\text{edge}}[v', u] + \frac{1}{m}$
return $\hat{c}_{\text{edge}}[u, v]$ for all pairs of vertices $u, v \in V$

The proof of Theorem 8 is analogous to Theorem 5, using Theorems 1 and 7 to derive the number of samples required.

Theorem 8. *Given a graph $G = (V, E)$, accuracy ϵ, and confidence $1 - \delta$, the estimates $\hat{c}_{edge}(u, v)$ obtained by Algorithm 2 satisfy*

$$\Pr\left(\forall u, v \in V,\ |c_{edge}(u, v) - \hat{c}_{edge}(u, v)| \leq \epsilon\right) \geq 1 - \delta.$$

Theorem 9. *Algorithm 2 has running time $\mathcal{O}(\Delta \lg \Delta + |E|)$.*

3.3 ESTIMATION VIA WEDGE NORMALIZATION

In this section, we present our strategy for estimating the wedge-normalized CN $c_{\text{wedge}}(u, v)$ for all $u, v \in V$.

3.3.1 Range Space and VC Dimension Results

Let $|W|$ be our sample space, and $E_{\text{wedge}}(u, v)$ the event of sampling a wedge in $N(u) \cap N(v)$, that is, $E_{\text{wedge}}(u, v) = \{(u, x, v) \mid x \in N(u) \cap N(v)\}$. Then,

$$\Pr\left(E_{\text{wedge}}(u, v)\right) = \frac{|N(u) \cap N(v)|}{|W|} = c_{\text{wedge}}(u, v).$$

We can now approximate $c_{\text{wedge}}(u, v)$ by estimating the probability of $E_{\text{wedge}}(u, v)$.

Let $(W, \mathcal{R}_{\text{wedge}})$ be a range space, where $\mathcal{R}_{\text{wedge}}$ is the set of all events $E_{\text{wedge}}(u, v)$ for each pair of vertices $u, v \in V$, i.e.,

$$\mathcal{R}_{\text{wedge}} = \left\{E_{\text{wedge}}(u, v) \mid u, v \in V\right\}.$$

Theorem 10. *The VC dimension of $(W, \mathcal{R}_{wedge})$ is $d_{VC}(\mathcal{R}_{wedge}) = 1$ if $|W| \geq 1$, and $d_{VC}(\mathcal{R}_{wedge}) = 0$ otherwise.*

Proof. If $|W| = 0$, then the range space is empty, and its VC dimension is 0. For $|W| \geq 1$, consider an arbitrary wedge (u, x, v). The set $\{(u, x, v)\}$ is shattered by $\mathcal{R}_{\text{wedge}}$, since $(u, x, v) \in E_{\text{wedge}}(u, v)$ and $(u, x, v) \notin E_{\text{wedge}}(u, x)$. Therefore, $d_{\text{VC}}(\mathcal{R}_{\text{wedge}}) \geq 1$. On the other hand, a wedge is linked to a single specific common neighborhood, which implies that the intersection between two events $E_{\text{wedge}}(u, v)$ and $E_{\text{wedge}}(u', v')$ is always empty. Thus, $\mathcal{R}_{\text{wedge}}$ cannot shatter sets of size 2 or greater. Therefore, $d_{\text{VC}}(\mathcal{R}_{\text{wedge}}) \leq 1$. Combining both inequalities, we conclude that $d_{\text{VC}}(\mathcal{R}_{\text{wedge}}) = 1$. □

3.3.2 Algorithm

Let $S = (s_1, s_2, \ldots, s_m) \in W^m$ be a i.i.d. sample of size m, where each s_i is a wedge in W. We use the relative frequency of $E_{\text{wedge}}(u, v)$ as an estimator for $c_{\text{wedge}}(u, v)$, i.e.,

$$\hat{c}_{\text{wedge}}(u, v) = \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}_{E_{\text{wedge}}(u,v)}(s_i), \tag{3.3}$$

where $\mathbb{1}_{E_{\text{wedge}}(u,v)}(s_i)$ is an indicator function that is equal to 1 if wedge $s_i \in N(u) \cap N(v)$, and 0 otherwise. Note that $\mathbb{E}\left[\hat{c}_{\text{wedge}}(u, v)\right] = c_{\text{wedge}}(u, v)$, i.e., $\hat{c}_{\text{wedge}}(u, v)$ is an unbiased estimator for $c_{\text{wedge}}(u, v)$.

Since the VC dimension does not depend on Δ, it is not necessary to call GETMAXIMUMDEGREE (as done by Algorithms 1 and 2), saving some preprocessing time.

Also, because each wedge corresponds to a one count of the neighborhood, for each sampled wedge u, w, v we update only the estimate $\hat{c}_{\text{wedge}}(u, v)$, which reduces the time complexity per sample to $\mathcal{O}(1)$. Algorithm 3 presents the pseudo-code of the wedge-normalized CN algorithm. The value for the constant b used in the evaluation of the sample size is discussed further in Section 5.

Algorithm 3: WEDGENORMALIZEDCN(G, ε, δ)

input : Graph $G = (V, E)$, accuracy ϵ, confidence $1 - \delta$
output : Estimates $\hat{c}_{\text{wedge}}(u, v)$ for all pairs of vertices $u, v \in V$
Initialize all $\hat{c}_{\text{wedge}}[u, v]$ to zero
Precompute alias table for efficient wedge sampling
$m \leftarrow \left\lceil \frac{b}{\epsilon^2} \left(1 + \ln\left(\frac{1}{\delta}\right)\right)\right\rceil$
for $i \leftarrow 1$ **to** m **do**
    $u, w, v \leftarrow$ SAMPLEWEDGE(G)
    $\hat{c}_{\text{wedge}}[u, v] \leftarrow \hat{c}_{\text{wedge}}[u, v] + \frac{1}{m}$
return $\hat{c}_{\text{wedge}}[u, v]$ for all pairs of vertices $u, v \in V$

Theorem 11. *Given a graph $G = (V, E)$, accuracy ϵ, and confidence $1 - \delta$, the estimates $\hat{c}_{wedge}(u, v)$ obtained by Algorithm 3 satisfy*

$$\Pr\left(\forall u, v \in V,\ |c_{wedge}(u, v) - \hat{c}_{wedge}(u, v)| \leq \epsilon\right) \geq 1 - \delta.$$

The proof of Theorem 11 is analogous to Theorem 5.

Now we state the time complexity of Algorithm 3. The details of the running time of the sampling algorithm will be discussed in the next session.

Theorem 12. *Algorithm 3 has running time $\mathcal{O}(|E|)$.*

Proof. The time complexity of Algorithm 3 is dominated by the time complexity of the initialization of the alias table, which is $\mathcal{O}(|E|)$, as shown by Theorem 14. Because it takes $\mathcal{O}(\Delta)$ time to sample a wedge (again, as shown by Theorem 14), the time complexity of the loop is $\mathcal{O}(m\Delta)$, where m is the number of samples. Since m is a constant, the overall time is $\mathcal{O}(|E|) + \mathcal{O}(\Delta) = \mathcal{O}(|E|)$. □

3.3.3 Wedge Sampling

We now present a procedure to uniformly sample a wedge from the graph. Let $E^* = \{(u, v), (v, u) \mid \{u, v\} \in E\}$ be the set of $2|E|$ ordered pairs induced by the edges of the graph. For each pair (u, v), define $p_{u,v} = \frac{\delta_v - 1}{2|W|}$ as the probability of sample a pair from E^*. Algorithm 4 describes how to sample a wedge from the graph.

In order to define the values of $p_{u,v}$, we need to know $|W|$. Proposition 1 provides a way to compute $|W|$ in $\mathcal{O}(|E|)$ time.

Theorems 13 and 14 states the correctness and time complexity of Algorithm 4, respectively.

Theorem 13. *Algorithm 3 selects a wedge uniformly at random from W.*

Proof. Fix a wedge (u, x, v). In the first step of the algorithm, the probability of sampling the ordered pair (u, x) is $\Pr((u, x)) = p_{u,x} = \frac{\delta_x - 1}{2|W|}$. Conditioned on sampling the pair (u, x),

Algorithm 4: SAMPLEWEDGE(G)

input : Graph $G = (V, E)$
output : A wedge uniformly sampled from G
Let $E^* = \{(u, v), (v, u) \mid \{u, v\} \in E\}$ be the set of $2|E|$ ordered pairs induced by the edges of G
Let $p_{u,v} = \frac{\delta_v - 1}{2|W|}$ be the probability of sampling the ordered pair (u, v)
1. Sample an ordered pair (u, w) from E^* with probability $p_{u,w}$
2. Sample a vertex v from $N(w) \setminus u$
3. Return the wedge u, w, v

the probability of sampling v (thus forming (u, x, v)) is $\Pr(v \mid (u, x)) = \frac{1}{|N(x) \setminus u|} = \frac{1}{\delta_x - 1}$. Alternatively, (u, x, v) can be formed by first sampling the pair (v, x) and then sampling u. The probability of sampling the wedge (u, x, v) is then

$$\begin{aligned}\Pr((u, x, v)) &= \Pr((u, x)) \cdot \Pr(v \mid (u, x)) + \Pr((v, x)) \cdot \Pr(u \mid (v, x)) \\ &= \frac{\delta_x - 1}{2|W|} \cdot \frac{1}{\delta_x - 1} + \frac{\delta_x - 1}{2|W|} \cdot \frac{1}{\delta_x - 1} = \frac{1}{|W|}.\end{aligned}$$

Therefore, the algorithm samples a wedge uniformly at random from W. Finally, observe that $p_{u,v}$ is a valid probability distribution, as

$$\sum_{(s,t) \in E^*} p_{s,t} = \sum_{\{s,t\} \in E} p_{s,t} + p_{t,s} = \sum_{\{s,t\} \in E} \frac{\delta_t - 1}{2|W|} + \frac{\delta_s - 1}{2|W|} = \sum_{\{s,t\} \in E} \frac{\delta_s + \delta_t - 2}{2|W|} = 1.$$

The last equality follows from Proposition 1. □

Theorem 14. *Algorithm 4 requires $\mathcal{O}(|E|)$ preprocessing time and samples a wedge in $\mathcal{O}(\Delta)$ time.*

Proof. To define the values of p_e, we need to compute $|W|$. This can be done in $\mathcal{O}(|E|)$ time using Proposition 1. To sample a pair (u, v) with probability $p_{u,v}$, we can use the Alias method (Walker, 1974), which takes $\mathcal{O}(|E|)$ time to preprocess the internal table with $2|E|$ pairs and $\mathcal{O}(1)$ time to sample a pair. Sampling of a vertex from $N(v) \setminus u$ takes $\mathcal{O}(\Delta)$ time, as we iterate through the neighbors of w and choose one vertex randomly. □

4 COMMON NEIGHBORS ESTIMATION

In this section, we show how the normalized versions can be used to obtain $c(u, v)$. First, in Section 4.1, we explain the necessity of using a (η, ϵ)-approximation (see Definition 7) and describe our denormalization strategy. We then present the algorithms and theorems concerning their approximation guarantees and running times. Subsequently, in Section 4.2, we show that the quality of the estimators employed by the sampling strategies can affect the resulting approximation. This gives rise to a trade-off between running time and estimator quality, which must be taken into account when selecting a strategy.

4.1 DENORMALIZATION STRATEGY

Consider the vertex-normalized estimator $\hat{c}_{\text{vertex}}(u, v)$ in Equation 3.1. Multiplying it by $|V|$ provides an estimator $\hat{c}(u, v)$ for $c(u, v)$, i.e.,

$$\hat{c}(u, v) = |V| \cdot \hat{c}_{\text{vertex}}(u, v).$$

Since $\hat{c}_{\text{vertex}}(u, v)$ is an unbiased estimator, $\hat{c}(u, v)$ inherits this property and is also unbiased.

A straightforward approach, but one that suffers from a major drawback, is to find an absolute ϵ-approximation, i.e,

$$|c_{\text{vertex}}(u, v) - \hat{c}_{\text{vertex}}(u, v)| \leq \epsilon, \text{ for all } u, v \in V.$$

By denormalizing, we obtain

$$|V| \cdot |c_{\text{vertex}}(u, v) - \hat{c}_{\text{vertex}}(u, v)| \leq |V| \cdot \epsilon,$$

which, by Definition 2, is equivalent to

$$|c(u, v) - \hat{c}(u, v)| \leq |V| \cdot \epsilon,$$

and therefore the error increases linearly with the size of the graph. To overcome this, we use a relative (p, ϵ)-approximation (Definition 7). When the threshold η satisfies certain conditions (e.g., is sufficiently small or appropriately chosen), we can derive the following bound

$$|c_{\text{vertex}}(u, v) - \hat{c}_{\text{vertex}}(u, v)| \leq c_{\text{vertex}}(u, v)\epsilon,$$

and, by denormalizing it,

$$|V| \cdot |c_{\text{vertex}}(u, v) - \hat{c}_{\text{vertex}}(u, v)| \leq |V| \cdot c_{\text{vertex}}(u, v)\epsilon.$$

Again, by Definition 2, this is equivalent to

$$|c(u, v) - \hat{c}(u, v)| \leq c(u, v)\epsilon.$$

Notice now that the error is relative to $c(u, v)$, instead of the graph size. An analogous approach to the one described above can be used to denormalize $c_{\text{edge}}(u, v)$ and $c_{\text{wedge}}(u, v)$, so the details are omitted.

We proceed to introduce the denormalized versions of Algorithms 1, 2, and 3. These algorithms differ in three ways: an additional input parameter η, a sample size computed according to Theorem 2 (which uses η), and denormalized output values (multiplied by $|V|$, $\frac{|E|}{2}$ or $|W|$). These modified algorithms are presented in Algorithms 5, 6, and 7.

Algorithm 5: $\mathrm{CN}(G, \varepsilon, \delta, \eta)$ (vertex sampling)

input: Graph $G = (V, E)$, accuracy ϵ, confidence $1 - \delta$, threshold η
output : Estimates $\hat{c}(u, v)$ for all pairs $u, v \in V$
$\Delta \leftarrow$ getMaximumDegree(G)
$m \leftarrow \left\lceil \frac{b'}{\epsilon^2 \eta} \left(\lfloor 2 \lg(\Delta) \rfloor \ln \frac{1}{\eta} + \ln \frac{1}{\delta} \right) \right\rceil$
for $i \leftarrow 1$ **to** m **do**
  $s \leftarrow$ sampleVertex(G)
  forall *pairs* $n_i, n_j \in N(s)$ **do**
    $\hat{c}_{\text{vertex}}[n_i, n_j] \leftarrow \hat{c}_{\text{vertex}}[n_i, n_j] + \frac{1}{m}$
return $\hat{c}[u, v] = |V| \cdot \hat{c}_{\text{vertex}}[u, v]$ for all pairs $u, v \in V$

Algorithm 6: $\mathrm{CN}(G, \varepsilon, \delta, \eta)$ (edge sampling)

input: Graph $G = (V, E)$, accuracy ϵ, confidence $1 - \delta$, threshold η
output : Estimates $\hat{c}(u, v)$ for all pairs $u, v \in V$
$\Delta \leftarrow$ getMaximumDegree(G)
$m \leftarrow \left\lceil \frac{b'}{\epsilon^2 \eta} \left((\lfloor \lg(\Delta) \rfloor + 2) \ln \frac{1}{\eta} + \ln \frac{1}{\delta} \right) \right\rceil$
for $i \leftarrow 1$ **to** m **do**
  $\{u, v\} \leftarrow$ sampleEdge(G)
  for *each* $u' \in N(u) \setminus \{v\}$ **do**
    $\hat{c}_{\text{edge}}[u', v] \leftarrow \hat{c}_{\text{edge}}[u', v] + \frac{1}{m}$
  for *each* $v' \in N(v) \setminus \{u\}$ **do**
    $\hat{c}_{\text{edge}}[v', u] \leftarrow \hat{c}_{\text{edge}}[v', u] + \frac{1}{m}$
return $\hat{c}[u, v] = \frac{|E|}{2} \cdot \hat{c}_{\text{edge}}[u, v]$ for all pairs $u, v \in V$

Algorithm 7: $\mathrm{CN}(G, \varepsilon, \delta, \eta)$ (wedge sampling)

input: Graph $G = (V, E)$, accuracy ϵ, confidence $1 - \delta$, threshold η
output : Estimates $\hat{c}(u, v)$ for all pairs $u, v \in V$
$m \leftarrow \left\lceil \frac{b'}{\epsilon^2 \eta} \left(\ln \frac{1}{\eta} + \ln \frac{1}{\delta} \right) \right\rceil$
for $i \leftarrow 1$ **to** m **do**
  $\{u, w, v\} \leftarrow$ sampleWedge(G)
  $\hat{c}_{\text{wedge}}[u, v] \leftarrow \hat{c}_{\text{wedge}}[u, v] + \frac{1}{m}$
return $\hat{c}[u, v] = |W| \cdot \hat{c}_{\text{wedge}}[u, v]$ for all pairs $u, v \in V$

While the role of ϵ and δ on the quality of the output is well understood, the impact of η is less evident. The selection of appropriate values for η is discussed in Section 5.

Theorems 15, 16, and 17 below summarize the results of the denormalization by vertices, edges, and wedges respectively.

Theorem 15. *Given a graph $G = (V, E)$, parameters ϵ, δ and η, Algorithm 5 has running time $\mathcal{O}(\Delta^2 \lg \Delta + |E|)$ and returns $\hat{c}(u, v)$ s.t.*

$$\Pr\left(\forall u, v \in V,\ |\hat{c}(u, v) - c(u, v)| \leq \epsilon \max\left\{c(u, v), |V|\eta\right\}\right) \geq 1 - \delta.$$

Theorem 16. *Given a graph $G = (V, E)$, parameters ϵ, δ and η, Algorithm 6 has running time $\mathcal{O}(\Delta \lg \Delta + |E|)$ and returns $\hat{c}(u, v)$ s.t.*

$$\Pr\left(\forall u, v \in V,\ |\hat{c}(u, v) - c(u, v)| \leq \epsilon \max\left\{c(u, v), \tfrac{|E|}{2}\eta\right\}\right) \geq 1 - \delta.$$

Theorem 17. *Given a graph $G = (V, E)$, parameters ϵ, δ and η, Algorithm 7 has running time $\mathcal{O}(|E|)$ and returns $\hat{c}(u, v)$ s.t.*

$$\Pr\left(\forall u, v \in V,\ |\hat{c}(u, v) - c(u, v)| \leq \epsilon \max\left\{c(u, v), |W|\eta\right\}\right) \geq 1 - \delta.$$

Although the wedge sampling strategy has the lowest running time compared to the other two variants, it also has the lowest accuracy. In Section 4.2), we discuss the reason behind this behavior, as well as compare the algorithms both in terms of time and accuracy.

4.2 RUNNING TIME VS ESTIMATION QUALITY TRADE-OFF

Observing only the time complexity of Algorithms 5, 6, and 7 ($\mathcal{O}(\Delta^2 \lg \Delta + |E|)$, $\mathcal{O}(\Delta \lg \Delta + |E|)$, and $\mathcal{O}(|E|)$, respectively), one might be tempted to conclude that the wedge sampling algorithm is always preferable. However, this conclusion overlooks the fact that the quality of the estimates produced by these algorithms can vary significantly.

For each sample, Algorithm 5 updates the estimates for $O(\Delta^2)$ vertex pairs, Algorithm 6 updates the estimates for $O(\Delta)$ pairs, and Algorithm 7 updates estimates for only a single pair. Compared to the other two algorithms, Algorithm 5 incorporates "more information" per sample, potentially leading to greater accuracy in its estimates. This suggests that the estimator of Algorithm 5 is more efficient than that of Algorithm 6, which in turn is more efficient than the estimator of Algorithm 7. This notion is formalized through the concept of *relative efficiency* between estimators (Wackerly et al., 2014).

Let $\hat{\theta}$ denote an estimator of θ. The *relative efficiency* $e(\hat{\theta}_1, \hat{\theta}_2)$ of two unbiased estimators $\hat{\theta}_1$ and $\hat{\theta}_2$ is a measure of how much more efficient $\hat{\theta}_1$ is compared to $\hat{\theta}_2$. If $e(\hat{\theta}_1, \hat{\theta}_2) > 1$, then $\hat{\theta}_1$ is considered more efficient than $\hat{\theta}_2$, which implies that $\hat{\theta}_1$ deviates less from the true value than $\hat{\theta}_2$. Formally, it is defined as

$$e(\hat{\theta}_1, \hat{\theta}_2) = \frac{\mathrm{Var}(\hat{\theta}_2)}{\mathrm{Var}(\hat{\theta}_1)}.$$

Theorem 18 presents results concerning the relative efficiency of the estimators of the previously presented algorithms.

Theorem 18. *Let $\hat{c}^{vertex}(u, v)$, $\hat{c}^{edge}(u, v)$ and $\hat{c}^{wedge}(u, v)$ be the estimates of $c(u, v)$ obtained, respectively, by Algorithms 5, 6, and 7. Then, for any pair of vertices u and v,*

(i) if $|E| > 2|V|$, then $\hat{c}^{vertex}(u, v)$ is more efficient than $\hat{c}^{edge}(u, v)$,

(ii) if $|E| < 2|V|$, then $\hat{c}^{edge}(u,v)$ is more efficient than $\hat{c}^{vertex}(u,v)$,

(iii) if $|E| = 2|V|$, then $\hat{c}^{vertex}(u,v)$ and $\hat{c}^{edge}(u,v)$ are equally efficient.

(iv) if $|W| > |E|$ and $|W| > |V|$, then $\hat{c}^{wedge}(u,v)$ is less efficient than $\hat{c}^{edge}(u,v)$ and $\hat{c}^{vertex}(u,v)$.

Proof. Let $E_{\text{vertex}}(u,v)$, $E_{\text{edge}}(u,v)$, and $E_{\text{wedge}}(u,v)$ be the events defined in Sections 3.1.1, 3.2.1, and 3.3.1, respectively. Then, by Equations 3.1, 3.2, and 3.3:

$$\hat{c}^{\text{vertex}}(u,v) = \frac{|V|}{m}\sum_{j=1}^{m} \mathbb{1}_{E_{\text{vertex}}(u,v)}(s_j), \quad \hat{c}^{\text{edge}}(u,v) = \frac{|E|}{2m}\sum_{j=1}^{m} \mathbb{1}_{E_{\text{edge}}(u,v)}(s_j),$$

$$\hat{c}^{\text{wedge}}(u,v) = \frac{|W|}{m}\sum_{j=1}^{m} \mathbb{1}_{E_{\text{wedge}}(u,v)}(s_j)$$

The variances of $\hat{c}^{\text{vertex}}(u,v)$, $\hat{c}^{\text{edge}}(u,v)$, and $\hat{c}^{\text{wedge}}(u,v)$ are given by

$$\begin{aligned}
\text{Var}(\hat{c}^{\text{vertex}}(u,v)) &= \frac{|V|^2}{m^2}\sum_{j=1}^{m} \text{Var}(\mathbb{1}_{E_{\text{vertex}}(u,v)}(s_j)) \\
&= \frac{|V|^2}{m^2}\sum_{j=1}^{m} c_{\text{vertex}}(u,v)(1 - c_{\text{vertex}}(u,v)) \\
&= \frac{|V|^2}{m^2}\sum_{j=1}^{m} \frac{c(u,v)(|V| - c(u,v))}{|V|^2} \\
&= \frac{c(u,v)(|V| - c(u,v))}{m}.
\end{aligned}$$

$$\begin{aligned}
\text{Var}(\hat{c}^{\text{edge}}(u,v)) &= \frac{|E|^2}{4m^2}\sum_{j=1}^{m} \text{Var}(\mathbb{1}_{E_{\text{edge}}}(s_j)) \\
&= \frac{|E|^2}{4m^2}\sum_{j=1}^{m} c_{\text{edge}}(u,v)(1 - c_{\text{edge}}(u,v)) \\
&= \frac{|E|^2}{4m^2}\sum_{j=1}^{m} \frac{2c(u,v)(|E| - 2c(u,v))}{|E|^2} \\
&= \frac{2c(u,v)(|E| - 2c(u,v))}{4m} \\
&= \frac{c(u,v)\left(\frac{|E|}{2} - c(u,v)\right)}{m}.
\end{aligned}$$

$$\begin{aligned}\mathrm{Var}(\hat{c}^{\mathrm{wedge}}(u,v)) &= \frac{|W|^2}{m^2}\sum_{j=1}^{m}\mathrm{Var}(\mathbb{1}_{E_{\mathrm{wedge}}(u,v)}(s_j)) \\ &= \frac{|W|^2}{m^2}\sum_{j=1}^{m} c_{\mathrm{wedge}}(u,v)(1-c_{\mathrm{wedge}}(u,v)) \\ &= \frac{|W|^2}{m^2}\sum_{j=1}^{m}\frac{c(u,v)(|W|-c(u,v))}{|W|^2} \\ &= \frac{c(u,v)(|W|-c(u,v))}{m}.\end{aligned}$$

The relative efficiency of $\hat{c}^{\mathrm{vertex}}(u,v)$ and $\hat{c}^{\mathrm{edge}}(u,v)$ is then

$$e(\hat{c}^{\mathrm{vertex}}(u,v),\hat{c}^{\mathrm{edge}}(u,v)) = \frac{\mathrm{Var}(\hat{c}^{\mathrm{edge}}(u,v))}{\mathrm{Var}(\hat{c}^{\mathrm{vertex}}(u,v))} = \frac{c(u,v)\left(\frac{|E|}{2}-c(u,v)\right)}{c(u,v)(|V|-c(u,v))} = \frac{\frac{|E|}{2}-c(u,v)}{|V|-c(u,v)}.$$

To proof (i), observe that if $|E| > 2|V|$, then $\frac{|E|}{2} - c(u,v) > |V| - c(u,v)$, which implies that $e(\hat{c}^{\mathrm{vertex}}(u,v),\hat{c}^{\mathrm{edge}}(u,v)) > 1$. Therefore, $\hat{c}^{\mathrm{vertex}}(u,v)$ is more efficient than $\hat{c}^{\mathrm{edge}}(u,v)$, regardless of the value of $c(u,v)$. The proof of (ii) and (iii) follow in a similar manner.

To prove (iv), observe that if $|W| > |E|$ and $|W > |V|$, we have that

$$e(\hat{c}^{\mathrm{wedge}}(u,v),\hat{c}^{\mathrm{edge}}(u,v)) = \frac{\mathrm{Var}(\hat{c}^{\mathrm{edge}}(u,v))}{\mathrm{Var}(\hat{c}^{\mathrm{wedge}}(u,v))} = \frac{\frac{|E|}{2}-c(u,v)}{|W|-c(u,v)} < 1.$$

$$e(\hat{c}^{\mathrm{wedge}}(u,v),\hat{c}^{\mathrm{vertex}}(u,v)) = \frac{\mathrm{Var}(\hat{c}^{\mathrm{vertex}}(u,v))}{\mathrm{Var}(\hat{c}^{\mathrm{wedge}}(u,v))} = \frac{|V|-c(u,v)}{|W|-c(u,v)} < 1.$$

These inequalities hold regardless of the value of $c(u,v)$. □

Except in cases of extreme sparsity (i.e., $|E| < 2|V|$), Theorem 18 implies that Algorithm 5 generally provides superior estimates to Algorithm 6 in practical applications. However, this improvement in accuracy comes at the cost of higher time complexity, creating a trade-off that must be considered when choosing between the two algorithms.

Regarding Algorithm 7, $|W|$ is generally larger than $|V|$ and $|E|$. For an example, consider a graph with $\Delta = 1000$. By Proposition 2, $|W|$ is at least $\binom{1000}{2} = 499500$; and by Proposition 1, $|E|$ is at least $1000 \cdot \frac{1000+1-2}{2} = 499500$. Since the size of $|W|$ grows quadratically with Δ, it is reasonable to expect that $|W|$ will be larger than $|E|$ and $|V|$ in most practical applications, so the estimator of the wedge sampling algorithm is overall less efficient than the other two.

5 EXPERIMENTAL EVALUATION

This section presents the results of our experimental evaluation. We implemented the algorithms in Python 3.10, using the NetworkX library for graph manipulation. We conducted the experiments on a machine with an AMD Ryzen 5700G CPU and 32GB of RAM, running Windows 11.

To establish a baseline for comparison, we computed the exact CN for all pairs using an adaptation of the current state-of-the-art algorithm by An et al. (An et al., 2019), which runs in $O(\Delta|E|)$. We maintain the iteration over all wedges, but instead of forming a wedge starting from the endpoints, we start from the central vertex. This minor modification bypasses an additional check done in the original algorithm to prevent counting the same wedge twice. For example, if we construct the wedge u, v, w from the endpoints, then there are two ways of forming this wedge: by starting from u, choosing v in $N(u)$ and then picking w from $N(v)$; or by starting from w, choosing v in $N(w)$ and then picking u from $N(v)$. Therefore, it is necessary to add a check to prevent counting the same wedge twice. However, if instead, we construct u, v, w from the central vertex v, then there is only one way of forming this wedge: by picking u and w from $N(v)$. Algorithm 8 presents the method used to compute the exact CN for all pairs.

Algorithm 8: ExactCN(G)

input : Graph $G = (V, E)$
output : $c(u, v)$ for all pairs of vertices $u, v \in V$
forall $v \in V$ **do**
  forall *pairs* $n_i, n_j \in N(v)$ **do**
    $c[n_i, n_j] \leftarrow c[n_i, n_j] + 1$
return $c[u, v]$ for all pairs of vertices $u, v \in V$

5.1 NORMALIZED CN ESTIMATION

This section evaluates Algorithms 1, 2, and 3 which approximate normalized CN. We used real-world graphs from the datasets SNAP (Leskovec and Krevl, 2014) and KONECT (Kunegis, 2013). Table 5.1 summarizes their key characteristics. Following the recommendation of Löffler and Phillips (Löffler and Phillips, 2009), we adopted $b = 1/2$.

Graph	Nodes	Edges	Wedges	Max. degree
Gowalla	196,591	950,327	290,400,040	14,730
Flickr links	1,715,254	15,551,250	14,660,775,060	27,224
Catster/dogster households	324,936	2,643,012	3,632,371,922	37,346
Youtube friendships	1,134,890	2,987,624	1,474,482,560	28,754
Hyves	1,402,673	2,777,419	1,447,128,030	31,883
Flixster	2,523,386	7,918,801	1,735,571,393	1,474

Table 5.1: Dataset details for real-world graphs.

In Figure 5.1, we report the running time of the algorithms on every graph for $\epsilon = 0.05$. The results were averaged ten times. It was not possible to directly run Algorithm

8 for the dataset since the output exceeded the system memory size. To remedy this problem, we saved the results of all pairs in the same key $c[0,0]$, instead of writing in the appropriate key $c[n_i, n_j]$. This modification produces a conservative estimate for the true running time of the exact algorithm. In our tests with smaller graphs (where it was possible to compute the running time directly), we observed that the estimate was 2 to 10 times lower than the actual value.

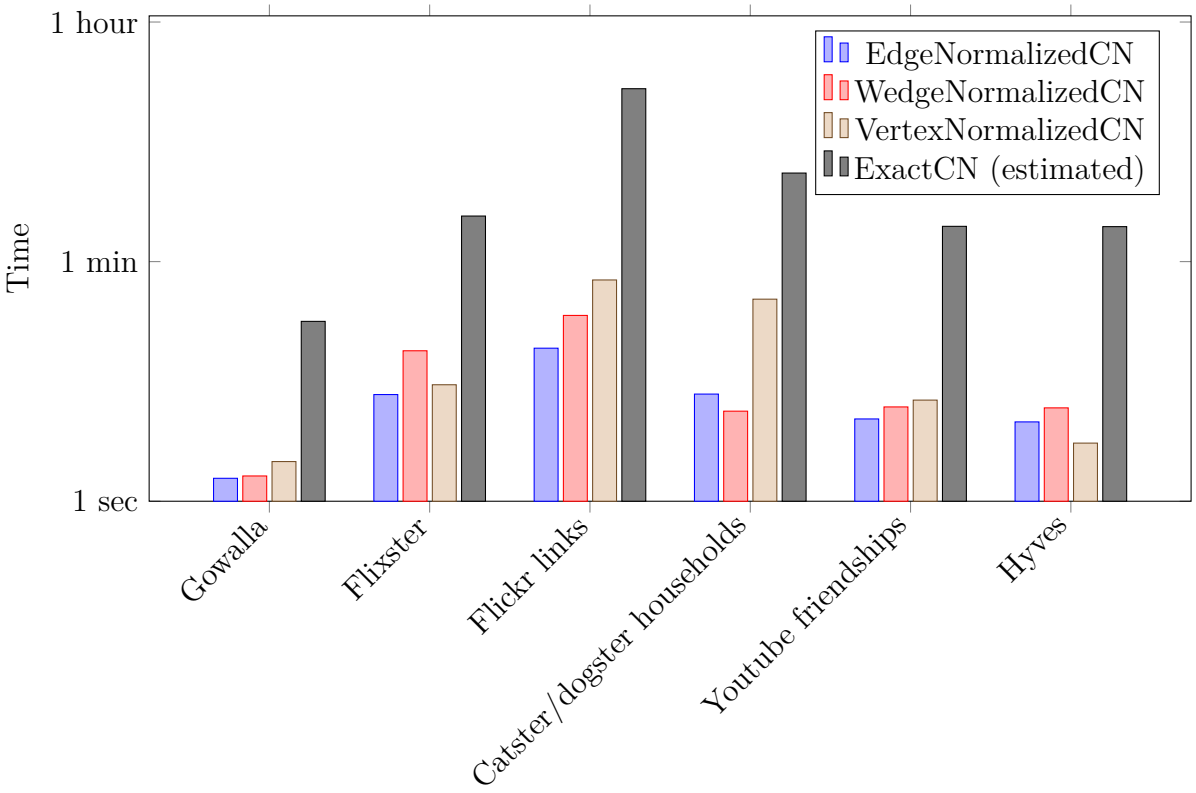


Figure 5.1: Comparison of running times for different algorithms, on all graphs in our dataset.

From the figure, we see that the probabilistic algorithms are much faster than the exact version, especially considering that our estimate for the ExactCN is at least 2 times lower than the actual value. Overall, the edge normalization algorithm is the fastest, except for graphs Catster/dogster households and Hyves. From the theoretical results, the wedge strategy should have been the fastest. This discrepancy is further examined in the next figures.

Figures 5.2 to 5.7 show the trade-off between accuracy and running time for the probabilistic algorithms. Accuracy was evaluated using the average of $|c_\bullet(u,v) - \hat{c}_\bullet(u,v)|$ over all non-empty intersections ($\bullet$ is "vertex", "edge" or "wedge"). Experiments were repeated ten times, and average results are reported for various ϵ in $[0.02, 0.04, 0.06, 0.08, 0.1]$ when possible. In some cases, due to lack of memory size, it was not possible to compute the results for $\epsilon = 0.02$ and/or $\epsilon = 0.04$.

From Figures 5.2 to 5.7, it is possible to see that the vertex strategy is the most accurate since its plot stays in the lower part of the figures. However, as expected, this results in a higher running time, as reflected by the stretch of its plot to the right. The edge normalization strategy has lower accuracy and running time, so its plot has a tendency to stay in the center or with a shift to the left. The behavior of the wedge strategy, however, is far from the theoretical prediction. According to theory, its plot should have been

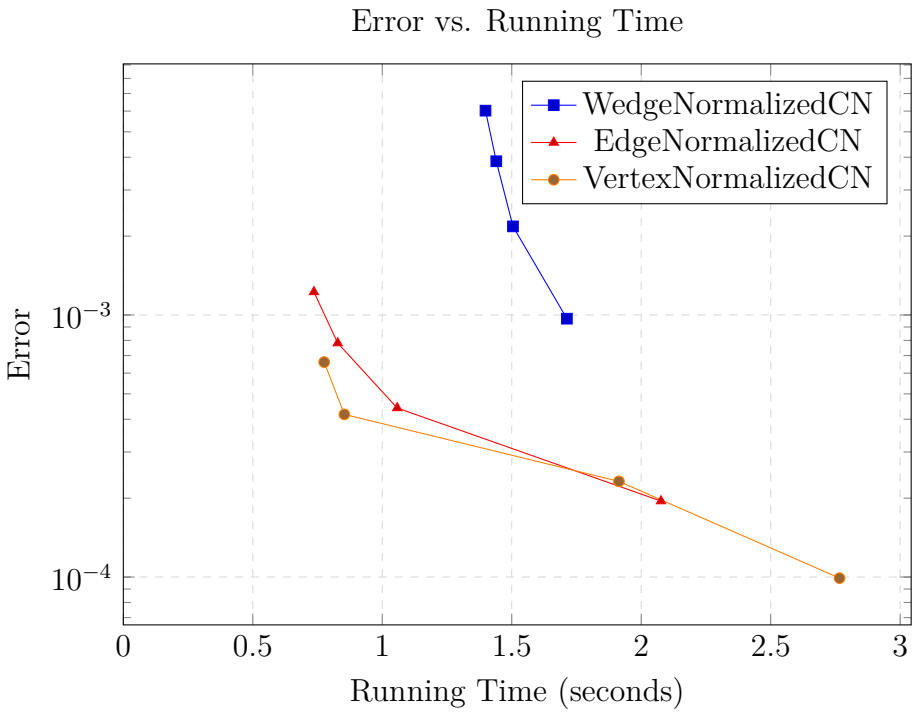


Figure 5.2: Error vs. Running Time comparison between the probabilistic algorithms for the Gowalla graph.

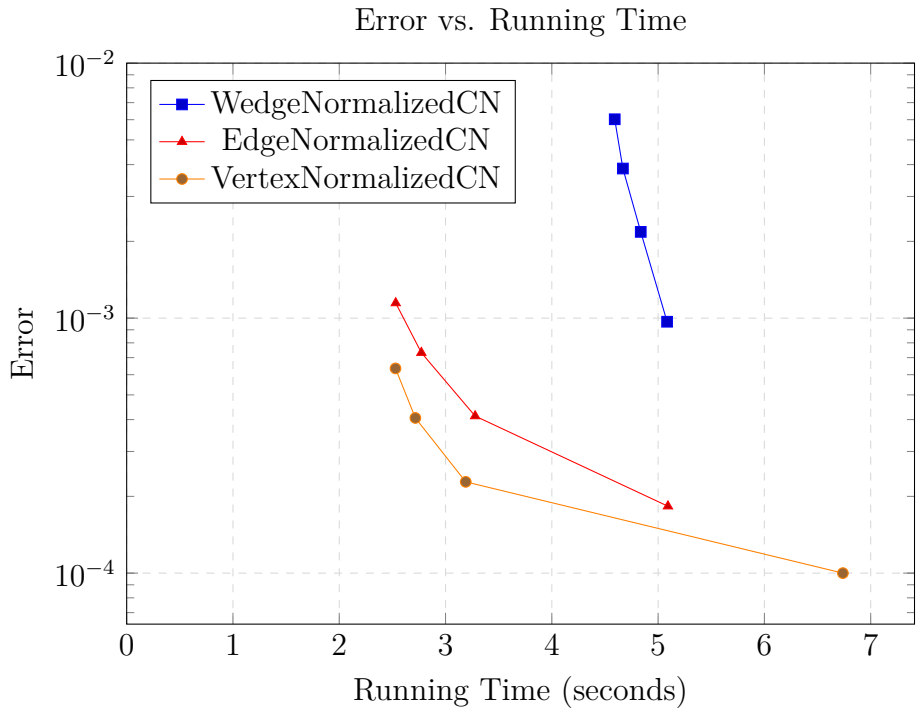


Figure 5.3: Error vs. Running Time comparison between the probabilistic algorithms for the Youtube friendships graph.

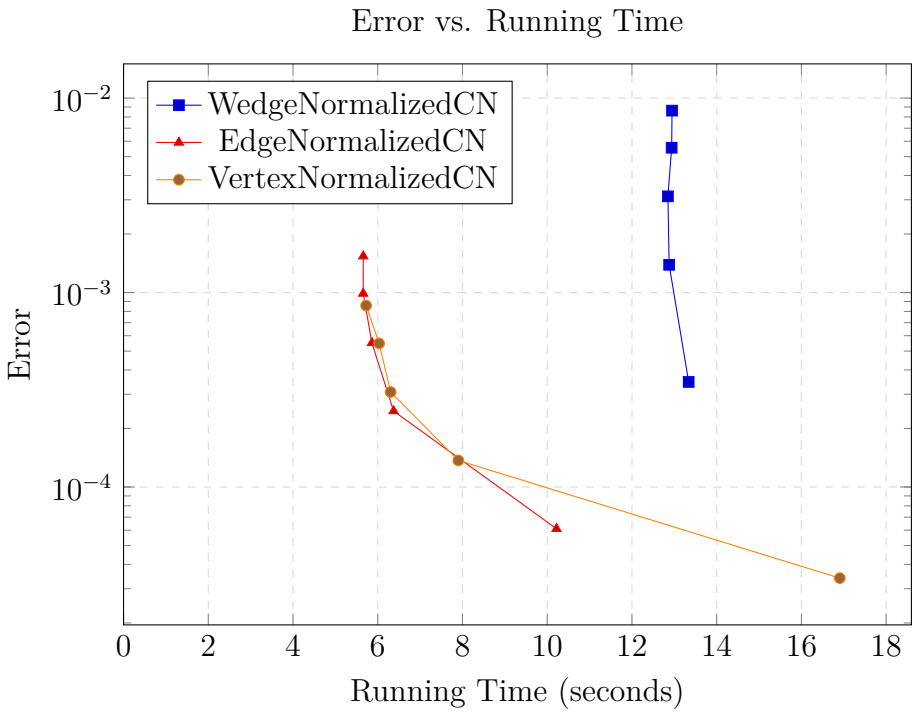


Figure 5.4: Error vs. Running Time comparison between the probabilistic algorithms for the Flixster graph.

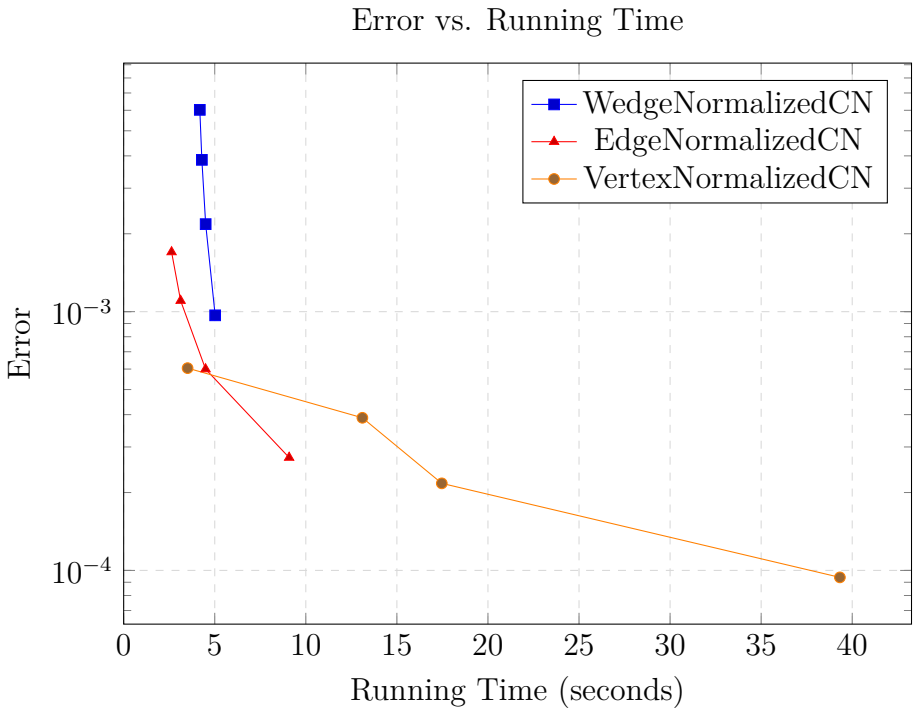


Figure 5.5: Error vs. Running Time comparison between the probabilistic algorithms for the Catster/dogster households graph.

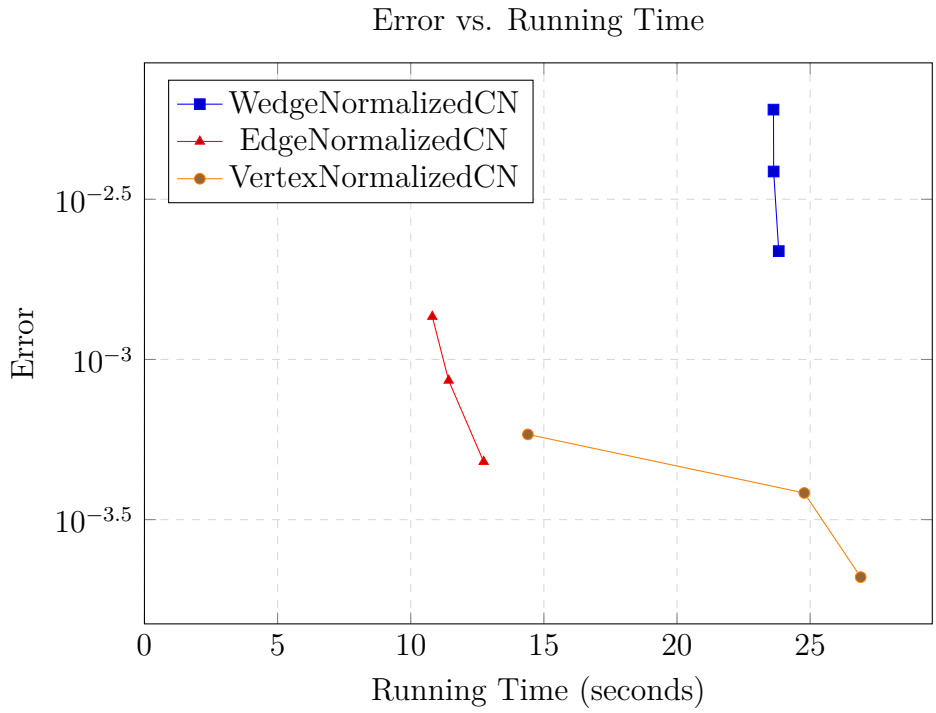


Figure 5.6: Error vs. Running Time comparison between the probabilistic algorithms for the Flickr links graph.

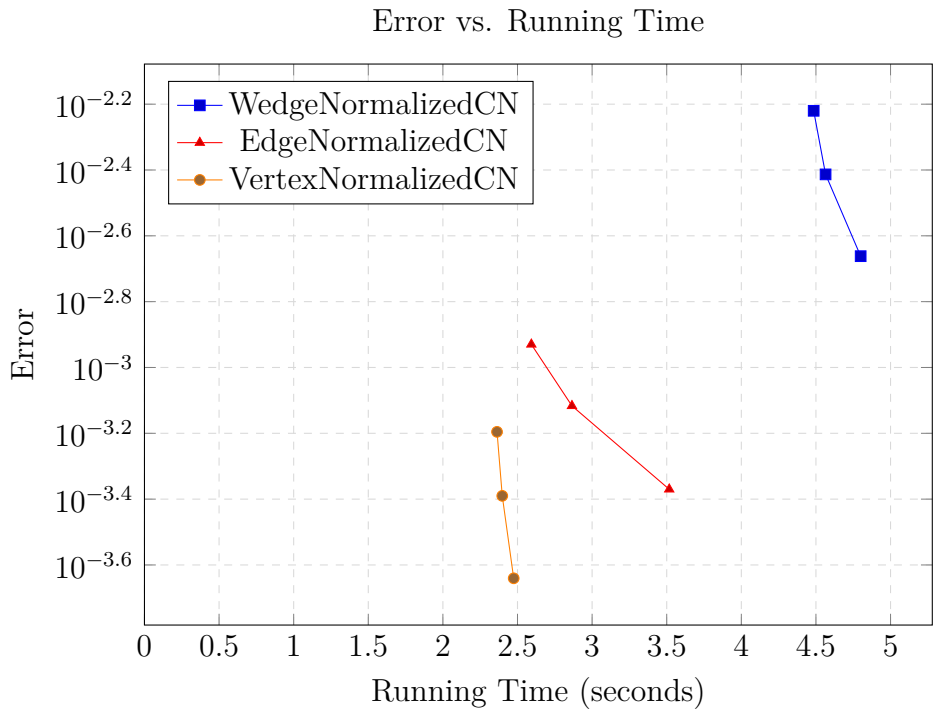


Figure 5.7: Error vs. Running Time comparison between the probabilistic algorithms for the Hyves graph.

positioned in the top left corner of the figures, as it has the least accuracy and the lowest running time of the three strategies.

The strange behavior of the wedge strategy can be explained by analyzing the preprocessing stage of the algorithm, specifically the steps taken by Algorithm 4. To construct the Alias table, which takes $O(|E|)$ time, it is necessary to know $|W|$, which also takes $O(|E|)$ time. Computing $|W|$, in turn, requires knowing the degrees of all vertices, a process that again takes $O(|E|)$. Therefore, it is necessary to iterate over all the edges of the graph three times. The other two strategies iterate only once over the edges - to compute the maximum degree of the graph. The main loop of the other two strategies is not slow enough to allow the wedge algorithm to take the lead. We confirmed this explanation when we performed code profiling on the wedge algorithm. The preprocessing time corresponded to more than 80% of the entire running time of the algorithm. This is also reflected in the figures. Since the preprocessing phase is not affected by ϵ and it is primarily responsible for the running time, it is not a surprise that the wedge algorithm has an almost entirely vertical shape.

Finally, to evaluate scalability, we generate ten Barabási-Albert graphs (Albert and Barabási, 2002). These are synthetic graphs generated through an incremental process where new nodes are added and connected to existing nodes with a probability proportional to the existing nodes' degrees (preferential attachment). They are widely used models for real-world networks due to their scale-free degree distribution. We used a number of vertices ranging from 125,000 to 2,000,000. We set $\delta = 0.1$ and $\epsilon = 0.05$. Average running times were computed across ten runs. Figure 5.8 and Figure 5.9 present the scalability results for Barabási-Albert graphs with 10 and 3 edges per new node, respectively.

The results from the figures support the previous analysis: as expected, the edge normalization algorithm is faster than the vertex normalization, and the wedge normalization algorithm took the highest time due to the preprocessing stage.

For the exact CN values, even for the smallest graph with $n = 125,000$ nodes, computation took 46 seconds with 10 edges per new node and 4 seconds with 3 edges per new node. For $n = 250,000$, the difference was even more significant: it took 111 seconds with 10 edges per new node and 10 seconds with 3 edges per new node. For n greater than $250,000$, it was not possible to compute the exact CN for graphs with 10 edges per new node, as the results exceeded memory size. For $n = 1,000,000$, computing the exact CN for a Barabási-Albert graph with 3 edges per new node took 49 seconds, which is significantly higher than all other values in Figure 5.9.

5.2 CN ESTIMATION

		Vertex Sampling (Algorithm 5)				Edge Sampling (Algorithm 6)			
p	Exact Alg. Time (s)	Mean Time (s)	Std. Dev. Time	Mean Rate (%)	Std. Dev. Rate	Mean Time (s)	Std. Dev. Time	Mean Rate (%)	Std. Dev. Rate
0.1	1.379	0.745	0.041	18.33%	0.005	2.814	0.123	30.73%	0.001
0.3	10.446	7.112	0.089	54.58%	0.010	9.135	0.141	52.39%	0.004
0.5	25.630	18.640	0.662	85.35%	0.008	16.085	1.113	66.9%	0.005
0.7	47.831	30.809	0.322	98.79%	0.003	21.215	0.207	74.86%	0.007
0.9	72.179	41.304	0.386	99.99%	0.000	26.873	0.434	80.77%	0.010

Table 5.2: Comparison of vertex and edge strategies for $G(n, p)$ graphs as a function of edges generation probabilities η.

This section evaluates Algorithms 5 and 6, which offer relative estimates for CN. We set $\epsilon = 0.1$, $\delta = 0.1$ and threshold $\eta = 0.7$ and $\eta = 0.04$ for the vertex and edge strategies, respectively, to balance runtime and accuracy. In the absence of specific

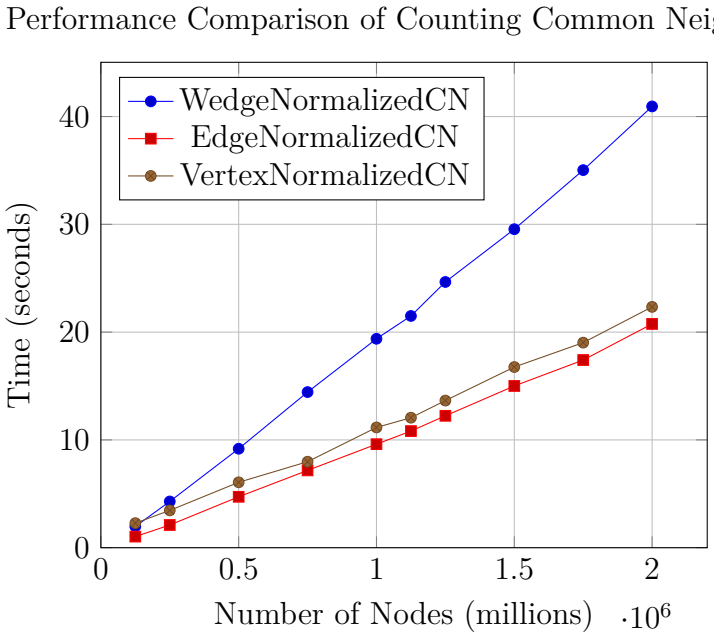


Figure 5.8: Running time of normalized algorithms on Barabási–Albert graphs as a function of graph size, with 10 edges per new node.

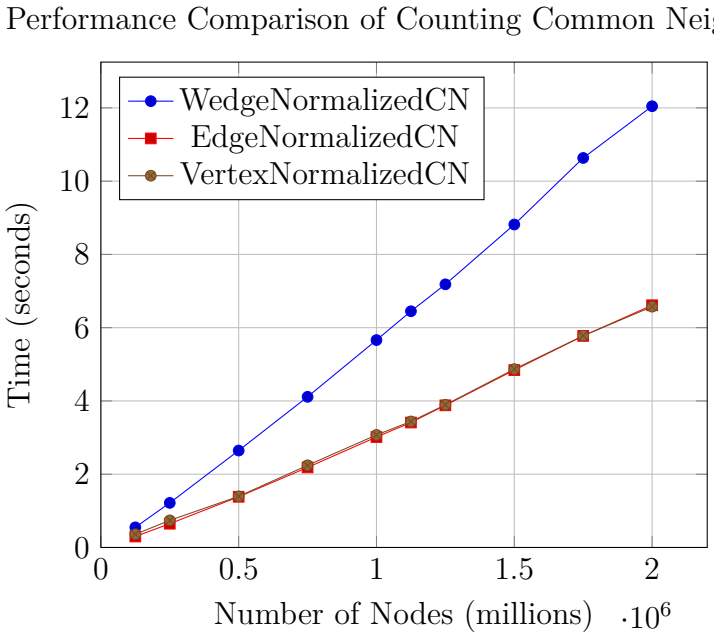


Figure 5.9: Running time of normalized algorithms on Barabási–Albert graphs as a function of graph size, with 3 edges per new node.

		Vertex Sampling (Algorithm 5)				Edge Sampling (Algorithm 6)			
n	Exact Alg. Time (s)	Mean Time (s)	Std. Dev. Time	Mean Rate (%)	Std. Dev. Rate	Mean Time (s)	Std. Dev. Time	Mean Rate (%)	Std. Dev. Rate
500	2.024	2.753	0.094	83.09%	0.009	4.444	0.077	81.15%	0.009
1000	23.104	18.115	0.267	84.85%	0.009	16.229	0.160	66.8%	0.007
2000	257.811	87.861	0.745	86.79%	0.005	41.976	0.167	52.44%	0.003
4000	2396.315	409.536	2.306	88.07%	0.007	97.159	0.357	40.01%	0.002

Table 5.3: Comparison of vertex and edge strategies for $G(n, p)$ graphs as a function of the number of vertices n.

recommendations, we set $b' = 1/2$, as we observed that this value is very satisfactory in practice. We assess the accuracy of our estimates by calculating the *average rate of successful approximations*. This rate is determined by dividing the number of non-empty neighborhood intersections that satisfy $|c(u,v) - \hat{c}(u,v)| \leq \epsilon c(u,v)$ by the total number of non-empty intersections. We also compared the runtime of our algorithms to that of the exact algorithm. Each experiment was performed ten times, and we took the average and standard deviation of the results.

Preliminary experiments on real-world graphs revealed either high running times or low rates of successful approximations for the algorithms, which we attribute primarily to the sparsity of the graphs. To investigate the impact of graph density, we generated synthetic graphs using the $G(n,p)$ model (Erdos and Renyi, 1959), a type of random graph where each pair of n vertices is connected by an edge with an independent probability p. We varied p to control the graph density and n to explore the effect of graph size.

Table 5.2 shows results for $G(n,p)$ graphs with $n = 1000$ and varying p. Denser graphs led to higher success rate for approximations (reported as "Mean Rate" and "Std. Dev. Rate" in the tables), and both our vertex and edge sampling algorithms outperformed the runtime of the exact algorithm. Table 5.3 shows results for $G(n,p)$ graphs with $p = 0.5$ and varying n. While faster, the edge strategy was less accurate than the vertex strategy. Notably, the accuracy of the vertex strategy improved with increasing n, whereas the accuracy of the edge strategy declined, suggesting better scalability for the vertex-based approach.

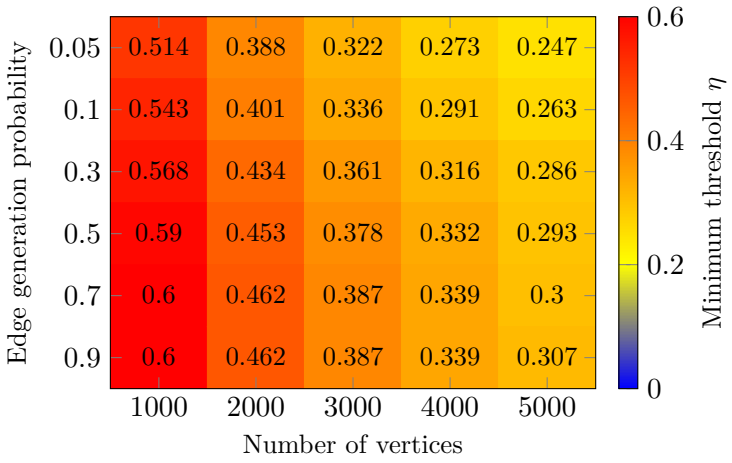
5.3 INFLUENCE OF THE THRESHOLD η

In this section, we study the influence of the threshold η in Algorithms 5 and 6. Theorem 2 establishes a relationship between the threshold η and sample size m. While a small η allows the approximation guarantees to be satisfied more often, it also increases sample size, impacting runtime. This trade-off means there is no single optimal η. However, a conceptual lower bound exists: when η is so small that the sample size exceeds the number of vertices or edges, performance is equivalent or worse than that of the exact solution. We refer to this lower bound as the *Minimum threshold* η. It can be calculated by setting $m = |V|$ or $m = |E|$ (depending on the algorithm used) in the equation of Theorem 2 and solving for η.

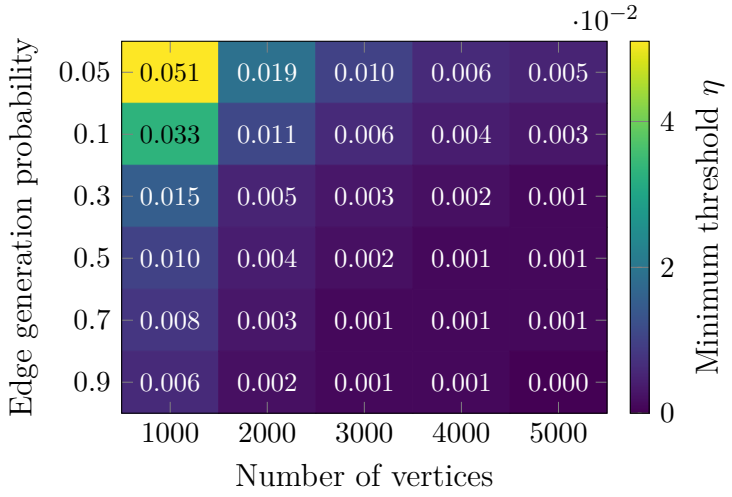
Note that the Minimum threshold η is determined by a formula that depends on the VC dimension, which, in turn, is a function of the maximum degree – a value that varies in random graphs. Consequently, we experimentally investigate the typical value of the Minimum threshold η. We generate $G(n,p)$ graphs, varying both n and p, and calculate the corresponding Minimum threshold η for each, using $\epsilon = 0.1$ and $\delta = 0.1$. Figures 5.10(a) and 5.10(b) illustrate the Minimum threshold η for Algorithms 5 and 6, respectively. The results are averaged over ten runs of the algorithms.

For the vertex strategy, the Minimum threshold η decreases with lower edge generation probabilities and larger graph sizes, allowing for smaller η values in sparser and larger graphs. For graphs with over 1000 vertices and an edge probability below 0.9, any $\eta \geq 0.6$ ensures an acceptable sample size. This justifies our choice of $\eta = 0.7$ for Algorithm 5, as it balances accuracy and runtime (see Tables 5.2 and 5.3). For the edge strategy, the Minimum threshold η exhibits a distinct behavior: it decreases as the edge generation probability increases, meaning denser, larger graphs allow for smaller η values

to improve accuracy. This explains our selection of $\eta = 0.04$ in our experiments with Algorithm 6.



(a) Minimum threshold η for Alg. 5 (Vertex strategy).



(b) Minimum threshold η for Alg. 6 (Edge strategy).

Figure 5.10: Minimum threshold η for vertex and edge strategies.

6 CONCLUSION

This paper introduces sampling-based algorithms for efficiently estimating CN for all vertex pairs in graphs. By leveraging VC dimension-based sample complexity analysis, we have designed algorithms with significantly improved performance compared to traditional methods. We present three distinct sampling strategies: vertex sampling, edge sampling and wedge sampling. Vertex sampling has a runtime of $O(\Delta^2 \lg \Delta + |E|)$ and offers higher estimator accuracy, while edge sampling runs faster at $O(\Delta \lg \Delta + |E|)$, and wedge sampling runs even faster at $O(|E|)$, where Δ is the maximum degree of the graph. This creates a trade-off between estimator quality and runtime.

Our experimental evaluation focused on denormalized and normalized versions of the algorithms, as well as the influence of the threshold η. The proposed strategies significantly outperformed exact methods in speed while exceeding quality guarantees. Interestingly, initial results suggest that denormalized strategies in real-world graphs often exhibit suboptimal performance compared to their normalized counterparts. Since many established metrics are based on normalized version of the metric, and our results confirm their superior performance, normalized versions may be of greater interest than denormalized ones.

Future research could investigate the use of adaptive sampling and explore alternative range spaces beyond vertex and edge sampling, provided that efficient sampling of the structures of interest is feasible.

REFERENCES

Albert, R. and Barabási, A.-L. (2002). Statistical mechanics of complex networks. *Rev. Mod. Phys.*, 74:47–97.

An, X., Gabert, K., Fox, J., Green, O., and Bader, D. A. (2019). Skip the intersection: Quickly counting common neighbors on shared-memory systems. In *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, pages 1–7.

Besta, M., Kanakagiri, R., Kwasniewski, G., Ausavarungnirun, R., Beránek, J., Kanellopoulos, K., Janda, K., Vonarburg-Shmaria, Z., Gianinazzi, L., Stefan, I., Luna, J. G., Golinowski, J., Copik, M., Kapp-Schwoerer, L., Di Girolamo, S., Blach, N., Konieczny, M., Mutlu, O., and Hoefler, T. (2021). Sisa: Set-centric instruction set architecture for graph mining on processing-in-memory systems. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, page 282–297, New York, NY, USA. Association for Computing Machinery.

Besta, M., Miglioli, C., Labini, P. S., Tětek, J., Iff, P., Kanakagiri, R., Ashkboos, S., Janda, K., Podstawski, M., Kwaśniewski, G., Gleinig, N., Vella, F., Mutlu, O., and Hoefler, T. (2022). Probgraph: high-performance and high-accuracy graph mining with probabilistic set representations. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, Dallas, Texas. IEEE Press.

Bloom, B. H. (1970). Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426.

Cousins, C., Wohlgemuth, C., and Riondato, M. (2023). Bavarian: Betweenness centrality approximation with variance-aware rademacher averages. *ACM Trans. Knowl. Discov. Data*, 17(6).

de Lima, A. M., da Silva, M. V., and Vignatti, A. L. (2022). Percolation centrality via rademacher complexity. *Discrete Applied Mathematics*, 323:201–216.

de Lima, A. M., da Silva, M. V. G., and Vignatti, A. L. (2020). Estimating the percolation centrality of large networks through pseudo-dimension theory. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1839–1847, New York, NY, USA. Association for Computing Machinery.

de Lima, A. M., da Silva, M. V. G., and Vignatti, A. L. (2022). Estimating the clustering coefficient using sample complexity analysis. In *LATIN 2022: Theoretical Informatics*, pages 328–341, Berlin, Heidelberg. Springer-Verlag.

Easley, D. and Kleinberg, J. (2010). *Networks, Crowds, and Markets: Reasoning about a Highly Connected World*. Cambridge University Press, USA.

Erdos, P. and Renyi, A. (1959). On random graphs i. *Publ. math. debrecen*, 6(290-297):18.

Har-Peled, S. and Sharir, M. (2011). Relative (p, ϵ)-approximations in geometry. *Discrete Comput. Geom.*, 45(3):462–496.

Kunegis, J. (2013). KONECT – The Koblenz Network Collection. In *Proc. Int. Conf. on World Wide Web Companion*, pages 1343–1350.

Leskovec, J. and Krevl, A. (2014). SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`.

Li, Y., Long, P. M., and Srinivasan, A. (2001). Improved bounds on the sample complexity of learning. *Journal of Computer and System Sciences*, 62(3):516 – 527.

Löffler, M. and Phillips, J. M. (2009). Shape fitting on point sets with probability distributions. In *Algorithms - ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*, pages 313–324. Springer Berlin Heidelberg, Berlin, Heidelberg.

Menczer, F., Fortunato, S., and Davis, C. (2020). *A First Course in Network Science*. Cambridge University Press, UK.

Mitzenmacher, M. and Upfal, E. (2017). *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis*. Cambridge University Press, UK, 2nd edition.

Newman, M. (2010). *Networks: An Introduction*. Oxford University Press, Inc., UK.

Pellegrina, L. and Vandin, F. (2023). Silvan: Estimating betweenness centralities with progressive sampling and non-uniform rademacher bounds. *ACM Trans. Knowl. Discov. Data*, 18(3).

Riondato, M. and Kornaropoulos, E. M. (2014). Fast approximation of betweenness centrality through sampling. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining*, page 413–422, New York, NY, USA. Association for Computing Machinery.

Riondato, M. and Kornaropoulos, E. M. (2016). Fast approximation of betweenness centrality through sampling. *Data Min. Knowl. Discov.*, 30(2):438–475.

Riondato, M. and Upfal, E. (2018). Abra: Approximating betweenness centrality in static and dynamic graphs with rademacher averages. *ACM Trans. Knowl. Discov. Data*, 12(5):61:1–61:38.

Shalev-Shwartz, S. and Ben-David, S. (2014). *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, UK.

Vapnik, V. N. and Chervonenkis, A. Y. (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability & Its Applications*, 16(2):264–280.

Wackerly, D., Mendenhall, W., and Scheaffer, R. L. (2014). *Mathematical Statistics with Applications*. Cengage Learning, USA.

Walker, A. (1974). New fast method for generating discrete random numbers with arbitrary frequency distributions. *Electronics Letters*, 10:127–128.