AGIS: Fast Approximate Graph Pattern Mining with Structure-Informed Sampling

Seoyong Lee
Seoul National University
Seoul, South Korea
sylee2685@snu.ac.kr

Jinho Lee
Seoul National University
Seoul, South Korea
leejinho@snu.ac.kr

ABSTRACT

Approximate Graph Pattern Mining (AGPM) is essential for analyzing large-scale graphs where exact counting is computationally prohibitive. While there exist numerous sampling-based AGPM systems, they all rely on uniform sampling and overlook the underlying probability distribution. This limitation restricts their scalability to a broader range of patterns.

In this paper, we introduce AGIS, an extremely fast AGPM system capable of counting arbitrary patterns from huge graphs. AGIS employs structure-informed neighbor sampling, a novel sampling technique that deviates from uniformness but allocates specific sampling probabilities based on the pattern structure. We first derive the ideal sampling distribution for AGPM and then present a practical method to approximate it. Furthermore, we develop a method that balances convergence speed and computational overhead, determining when to use the approximated distribution.

Experimental results demonstrate that AGIS significantly outperforms the state-of-the-art AGPM system, achieving $28.5\times$ geometric mean speedup and more than 100,000× speedup in specific cases. Furthermore, AGIS is the only AGPM system that scales to graphs with tens of billions of edges and robustly handles diverse patterns, successfully providing accurate estimates within seconds. We will open-source AGIS to encourage further research in this field.

PVLDB Reference Format:
Seoyong Lee and Jinho Lee. AGIS: Fast Approximate Graph Pattern Mining with Structure-Informed Sampling. PVLDB, 19(2): 238 - 251, 2025.
doi:10.14778/3773749.3773761

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/syleeKR/AGIS.

1 INTRODUCTION

Graph pattern mining (GPM) [1, 24, 32, 44, 51, 71, 77, 82] is one of the most time-consuming applications within graph processing workloads. Given a large data graph and a relatively small pattern graph, GPM searches for embeddings of the pattern from the data graph. Explicit pattern counts provide concise, deterministic summaries of higher-order graph structure, serving as fundamental descriptive statistics that underpin rigorous statistical analysis [49].

Proceedings of the VLDB Endowment, Vol. 19, No. 2 ISSN 2150-8097.
doi:10.14778/3773749.3773761

Moreover, as graphs are ubiquitous, pattern mining is employed across diverse domains, including bioinformatics [28, 38, 52], social network analysis [39, 87], and chemical compound classification [31]. More importantly, pattern counts are essential for interpretability and accountability in high-stakes settings, such as fraud detection and cybersecurity, where auditability and regulatory compliance depend on precise, reproducible evidence [3, 6, 27, 29]. Despite its significance in numerous real world scenarios, GPM suffers from its inherent high computational complexity, making it hard to scale to large graphs or complex patterns. This challenge is especially pertinent in the current era, as modern applications demand low-latency analysis on massive graphs [6, 27, 65].

One popular approach to alleviate this is approximate graph pattern mining (AGPM). Using the fact that GPM tasks, such as frequent subgraph mining [86] and motif counting [53], are employed in applications where estimated embedding counts suffice, AGPM computes approximate embedding counts for a given pattern within specified error bounds and confidence intervals. While there exist several seminal AGPM systems [7, 43, 91], they still have difficulties as the data graphs and patterns become larger. Upon characterizing them, we identify that certain patterns experience especially larger slowdowns. Our analysis shows that this can be attributed to the scale-free distribution [10, 11, 30] often found in real-world graphs. Such a distribution leads to a large variance in the embedding count depending on which part of the data graph is sampled. Because it makes the estimated count difficult to converge, this results in a long execution time for AGPM systems.

Fortunately, we identify that such a problem can be alleviated by assigning different probabilities to each sampling candidate. While existing methods also try to reduce the estimation variances, they often focus on narrowing down the candidates by only considering neighbors of already sampled vertices [43], or neighbors that satisfy certain constraints [7]. However, once the candidates set is constructed, the probability of each candidate to be sampled is uniform.

Instead of applying naive uniform sampling probability distribution, we propose to use uneven distribution where the probability to sample each candidate vertex is proportional to the number of potential embeddings found by choosing the vertex. In theory, this leads to a correct count without variance with a single sampler. However, this requires prior knowledge about the true embedding count, which defeats the purpose of AGPM. To construct an appropriate sampling probability distribution without relying on the true embedding count, we develop a method to approximate the distribution based on the structural characteristics of the data and pattern graph. By carefully examining the nearby structures of the

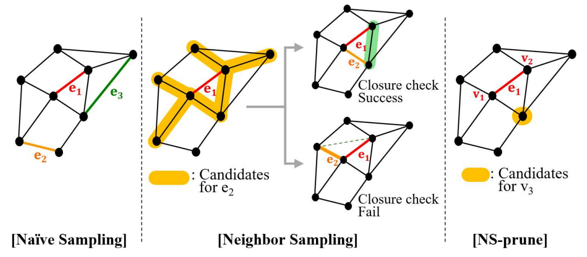


Figure 1: Sampling strategies of existing AGPM systems.

target vertex to be sampled, we calculate a distribution close to the ideal one with a small overhead.

Based on this, we propose AGIS, a fast and scalable AGPM system. At the heart of AGIS is *structure-informed neighbor sampling*. Structure-informed neighbor sampling leverages the calculated distribution and strategically determines when to apply it, thereby achieving an optimal balance between faster convergence and minimal computational overhead. In addition, we provide a heuristic to construct a matching order such that the benefit of structure-informed neighbor sampling is maximized. We evaluate AGIS over a various data graph and patterns, against several state-of-the-art baselines. Experimental results show that AGIS outperforms all the baselines, sometimes by an order of several magnitudes. AGIS sets a new state-of-the-art, further extending the applicability of AGPM methodology.

Our contributions can be summarized as follows:

- We show that by assigning different probabilities to each vertex in the sampling candidates, significantly faster convergence can be obtained for AGPM.
- We develop a method for constructing an approximate ideal sampling distribution and a heuristic decision process that determines when to apply it.
- We build AGIS, a fast AGPM system that significantly outperforms prior art over a diverse set of experiments.

2 BACKGROUND

In this section, we provide a comprehensive background on approximate graph mining and its core strategies in existing work. We use $G = (V_G, E_G)$ and $P = (V_P, E_P)$ to represent the data and pattern graphs, respectively. We assume that both graphs are undirected; thus, E_G and E_P are sets of unordered pairs $\{a, b\}$. For a vertex v, $d(v)$ denotes its degree, $\mathcal{N}(v)$ its set of neighbor vertices, and v.id its id. We use v for vertices in G and u for vertices in P.

2.1 Approximate Graph Mining Systems

Graph Pattern Mining. Graph Pattern Mining (GPM) is the problem of finding all *embeddings*, i.e., matches, of a pattern P within a given graph G. Formally, we define an embedding as a subgraph isomorphism from P to G. Specifically, it is a one-to-one mapping $\mathcal{M} : V_P \to V_G$ such that if $(u_i, u_j) \in E_P$, then $(\mathcal{M}(u_i), \mathcal{M}(u_j)) \in E_G$. We let $C(G, P)$ denote the total number of such embeddings. Example tasks in GPM include subgraph counting [32], subgraph listing (subgraph matching) [32], motif counting [53], and frequent subgraph mining [86].

Approximate Graph Pattern Mining. Approximate Graph Pattern Mining (AGPM) addresses the subgraph counting problem in an approximate manner. Given a graph G, a pattern P, an error bound ϵ, and a confidence level $1-\delta$, AGPM seeks to return a $(1 \pm \epsilon)$ approximation of the exact number of embeddings $C(G, P)$, with probability at least $1 - \delta$. AGPM systems enable users to mine arbitrary patterns using two major components: (1) a general sampling method applicable to any pattern, and (2) a convergence detection method that determines when to terminate sampling given (ϵ, δ).

2.2 Sampling Strategies of AGPM

To obtain approximate values for $C(G, P)$, existing approaches [7, 43, 91] employ the sampling method. As a simple example, suppose we were to approximate the number of triangles in graph G. In a straightforward `Naive` sampling method shown in Figure 1, a sampler randomly samples three edges e_1, e_2, e_3 and checks if they form a triangle. If they do form a triangle, the sampler estimates the number of triangles as $|E_G|^3$ since each of the three edges is sampled randomly with a probability of $\frac{1}{|E_G|}$. Otherwise, the sampler estimates the number of triangles as 0. The average from many such samplers will converge toward the true triangle count.

One significant issue with the `Naive` sampling method is its large variance between the sampler outputs. Because each sampler outputs either 0 or the substantially large value $|E_G|^3$, the average of samplers converges slowly. Neighbor Sampling (`NS`) [59] addresses this problem by leveraging the connectivity of the pattern graph. Specifically, a `NS` sampler operates as follows: it first samples an edge e_1 uniformly at random from E_G, similar to the sampler using `Naive` method. However, instead of sampling the next edge uniformly from E_G, it samples e_2 from the set of edges adjacent to e_1. Then, it performs a *closure check*, which determines whether an edge e_3 exists between the non-shared vertices of e_1 and e_2. This procedure confirms whether a triangle embedding can be formed from the sampled edges. The probability is $\frac{1}{|E_G| \cdot |N(e_1)|}$, where $N(e_1)$ denotes the set of neighboring edges of e_1. Because $|E_G| \cdot |N(e_1)|$ is smaller and closer to the true triangle count compared to $|E_G|^3$, the `NS` sampler outputs have lower variance and converges faster.

The aforementioned methods can easily be generalized to an arbitrary pattern P. A sampler sequentially samples vertices until the number of sampled vertices reaches $|V_P|$. Then, the probability p of this sampling sequence is given by

$$p = \prod_{i=1}^{|V_P|} Pr(v_i | v_1, \ldots, v_{i-1}),$$

where $Pr(v_i | v_1, \ldots, v_{i-1})$ denotes the probability to sample v_i given $v_1, \ldots v_{i-1}$ is sampled. Each sampler returns $\frac{1}{p}$ if the vertices form an embedding of P and returns 0 otherwise. This sampler serves as an unbiased estimator for $C(G, P)$ [43].

The `NS` method can further be modified by imposing restrictions on neighbor selection [19, 45]. For example, when sampling the `k-star` pattern (a pattern where k vertices are connected to a central vertex), we can sample the next edge only from the neighbors of the central vertex to maximize the success rate of the closure check. Building on this idea, ScaleGPM [7] proposes a modified version of `NS` called `NS-prune`. This method not only exploits the connectivity of the pattern P but also leverages its specific topology. It introduces two concepts of exact GPM systems:

Matching Order. A matching order [24, 36] $\pi : \{1, \ldots, |V_P|\} \to V_P$ is a permutation of the pattern vertices that specifies the order in which the vertices of P are matched to the graph G. We denote $u_i = \pi(i) \in V_P$ as the i-th vertex of P in the matching order. Correspondingly, we write $v_i \in V_G$ as the i-th sampled vertex of G.

Restriction Set. Given a matching order π, a restriction set [50] $\mathcal{R}^\pi$ is a set of ordered pairs of pattern vertices (u_i, u_j). $\mathcal{R}^\pi$ is applied with *symmetry breaking* [50], which enforces the condition v_i.id $< v_j$.id for all $(u_i, u_j) \in \mathcal{R}^\pi$. This is beneficial for exact mining systems, as it reduces the search space by exactly the number of automorphisms of P. Although this approach does not enumerate all embeddings, the total count can be recovered by simply multiplying the result accordingly.

`NS-prune` utilizes these concepts as follows. Taking the triangle pattern again as an example, it first samples $e_1 = \{v_1, v_2\}$ randomly like the prior methods. Then, instead of sampling any e_2 from the neighbors, it leverages the fact that u_3 is connected to u_1 and u_2. Thus, the next vertex v_3 must be selected from the intersection of the neighbors of v_1 and v_2, that is, $v_3 \in \mathcal{N}(v_1) \cap \mathcal{N}(v_2)$. By enforcing these *connectivity constraints* at each step, the success rate of finding an embedding increases, leading to faster convergence. Additionally, the triangle pattern has 3! number of automorphisms, having a restriction set $\mathcal{R}^\pi = \{(u_2, u_1), (u_3, u_2), (u_3, u_1)\}$. By applying symmetry breaking, the condition v_3.id $< \min(v_1.\text{id}, v_2.\text{id})$ is enforced. The enforcement of these restrictions reduces the set size of possible vertices. Since the sampling probability is equal to the inverse of the set size, the estimator's variance is reduced. Lastly, for general patterns, the `NS-prune` avoids sampling vertices from the previously sampled vertices.

Considering the above techniques, `NS-prune` samples the last vertex v_3 uniformly from the candidate set

$$S = \{v \mid v \in \mathcal{N}(v_1) \cap \mathcal{N}(v_2),\ v.id < \min(v_1.id,\ v_2.id)\}\,.$$

It then returns $|E_G| \cdot |S|$ as the sampler's output. In ScaleGPM, this value serves as an unbiased estimate of $C(G, P)$ divided by the number of automorphisms of the pattern, due to the use of the restriction set. Since the sampler produces non-zero outputs more frequently, its variance is reduced, leading to faster convergence.

2.3 Convergence Method

An important issue to be addressed from AGPM is to ensure convergence of $C(G, P)$ within the error bound ϵ. ASAP [43] and Arya [91] utilize the error-latency profile (ELP) heuristic that predetermines the number of samplers needed for convergence based on concentration inequalities. For example, Arya uses Chebyshev's inequality [8] to obtain the number of samplers as follows:

$$\#\text{ samplers needed} = \frac{K \times |E_G|^\rho}{C(G, P) \times \epsilon^2 \times \delta},$$

where ρ represents a pattern specific value, and K is a constant: However, the equation requires the value of $C(G, P)$, where the constant K is also unknown. To address this, they rely on sampling from a smaller subgraph of G to estimate K and $C(G, P)$.

Unfortunately, the method based on ELP lacks a theoretical guarantee as it relies on a subgraph whose K and $C(G, P)$ will be different from that of the original G. To address this problem, ScaleGPM [7] proposes a mathematically sound method to determine the termination of the sampling, as described in Algorithm 1 where Φ is the CDF of the standard normal distribution. Instead of predetermining the number of samplers, it detects convergence by using the mean and variance of the sampler results. The method is proven for arbitrary sampling-based techniques that provide unbiased estimates.

Algorithm 1 Online Convergence Detection

```
1: procedure CONVERGED($\epsilon$, $\delta$, L)
2:     $N \leftarrow |\mathbf{L}|$                          ▷ L is the list of sampler outputs
3:     $\mu \leftarrow \frac{1}{N}\sum_{i=1}^{N} L_i$,   $\sigma^2 \leftarrow \frac{1}{N}\sum_{i=1}^{N} L_i^2 - \mu^2$
4:     $\hat{\epsilon} \leftarrow \Phi^{-1}(1 - \frac{\delta}{2}) \cdot \frac{\sigma}{\sqrt{N}} \cdot \frac{1}{\mu}$            ▷ Estimated error
5:     return $\hat{\epsilon} \le \epsilon$
6: end procedure
```

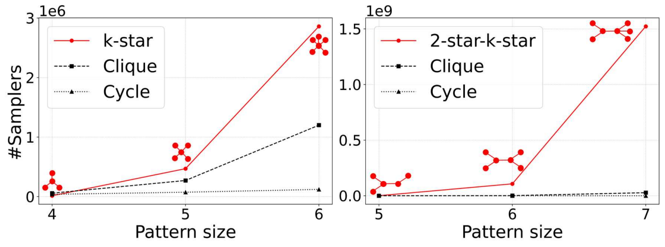


Figure 2: Number of samplers needed for convergence. LiveJournal graph, using $\epsilon = 0.1$, $\delta = 0.01$.

3 MOTIVATION

3.1 Limitations of Existing Methods

Even though AGPM systems achieve significantly shorter execution times compared to exact GPM systems, they are known to suffer from low convergence speed as the pattern size increases. Specifically, we find that the severity of such a problem is exacerbated by certain types of patterns. One example is patterns with many bridges [33], which are defined as edges whose removal would result in the separation of the graph. Figure 2 illustrates the number of samplers required for convergence on two patterns—`k-star` and `2-star-k-star`—on Livejournal [9] graph using ScaleGPM. Compared to the clique and cycle patterns with the same number of vertices, the two patterns we examine require more samplers to converge, with much steeper growth rates. Considering that a clique has far more edges than the examined patterns at an equal number of vertices, the results indicate that there exists some inefficiencies on top of pure complexity growth.

One reason for this is the scale-free distribution [10, 11, 30] observed in many real-world graphs, where the degree distribution of vertices is highly skewed. Suppose we are sampling the i-th vertex v_i, following a bridge $\{u_j, u_i\} \in E_p$ (with $j < i$). Accordingly, the connectivity constraints from Section 2.2 cannot be applied because, by the definition of a bridge, no other vertex u with $\pi^{-1}(u) < i$, except u_j, shares an edge with u_i. Therefore, omitting symmetry breaking for simplicity, $|S|$ is approximately equal to $d(v_j)$, the degree of v_j. Since the degrees vary significantly in graphs with a scale-free distribution, the sampler's outputs will also have a high variance in the estimate and ultimately slower convergence. Moreover, if vertex u_j in the pattern is connected to k bridges, the sampler's output will be roughly proportional to $d(v_j)^{k-1}$ (see Section 3.2), which exacerbates this issue.

3.2 Rethinking Sampling Probability

In the three sampling methods described in Section 2.2 (`Naive`, `NS`, and `NS-prune`) improvements were made by reducing the size of the sampling set. With a more promising candidate set, the probability of finding an embedding increases, leading to faster convergence.

These three methods differ in how they form a candidate set S, but they all assign uniform sampling probability to elements within S. In the `Naive` sampling method, the candidate set is the entire edge set E_G with the uniform sampling probability. In `NS`, the candidate set is narrowed to the edges neighboring the already-sampled vertices. For `NS-prune`, the candidate set is further refined only to the vertices that satisfy the connectivity constraints and symmetry-breaking restrictions. Assigning uniform probabilities to these set elements is a reasonable choice. However, such a choice is not mandatory. In fact, these methods can already be interpreted as assigning non-uniform probabilities restricted to the first vertex. All three methods uniformly sample a random edge at the beginning which can be interpreted as having all vertices in the set and having probability proportional to their degrees.

From these observations, we propose to further add an extra step to calculate a tailored sampling distribution, allowing us to assign distinct probabilities to each element in a way that it achieves faster convergence. As a motivational example, consider the `k-star` pattern. Suppose we use `NS-prune` without symmetry breaking and start with the central vertex. Since sampling the first edge uniformly is equivalent to sampling the first vertex with a probability proportional to its degree, a central vertex v is sampled with probability $\frac{d(v)}{2|E_G|}$. Subsequently, for the remaining k outer vertices, the candidate set is defined as the set of all neighbors of the central vertex, excluding any vertices that have already been sampled. Thus the sampler outputs $\frac{2|E_G|}{d(v)} \times d(v) \times (d(v)-1) \times \cdots \times (d(v)-(k-1))$. Since $d(v)$ varies greatly per vertex, it would need numerous iterations to achieve convergence.

On the other hand, an interesting aspect of `k-star` is that a vertex v can form exactly $\binom{d(v)}{k}$ such patterns. What if we discard uniform sampling and instead sample the first vertex v with probability proportional to $\binom{d(v)}{k}$; that is, $p = \frac{\binom{d(v)}{k}}{\sum_{x \in V_G} \binom{d(x)}{k}}$? The sampler's output will then be $\frac{\sum_{x \in V_G} \binom{d(x)}{k}}{\binom{d(v)}{k}} \times d(v) \times (d(v)-1) \times \cdots \times (d(v)-(k-1)) = \sum_{x \in V_G} \binom{d(x)}{k} \times k!$, which is a constant value equal to $C(G,P)$. This means that our sampler will always return the same value, resulting in zero variance, and convergence to $C(G,P)$ will occur immediately with a *single* sampler.

4 BUILDING SAMPLING DISTRIBUTIONS

While the result from Section 3.2 is highly appealing, two key challenges arise when applying the principle to general patterns. First, while identifying the ideal distribution for a simple pattern like `k-star` is straightforward—we can precompute the sampler outputs exactly—it is unclear how to find such a distribution for a general pattern. Second, even if we could determine the ideal distribution for a complex pattern, computing it directly could be prohibitively expensive, diminishing the advantages of faster convergence. In this section, we first discuss an ideal distribution for a general pattern (Section 4.1), how to approximate it (Section 4.2), the theoretical intuition behind it (Section 4.3) and how to preprocess the data graphs accordingly (Section 4.4).

4.1 Ideal Distribution for General Patterns

In this section, we first provide the ideal sampling probability distribution for a generalized pattern.

Definition 1 (Sampling Trajectory). *Given a graph G, a* sampling trajectory *τ is an ordered sequence of vertices sampled from V_G, denoted by*

$$\tau = (v_1, v_2, \ldots, v_k),$$

where $v_i \in V_G$ is the i-th sampled vertex.

If we sample a vertex $v \in V_G$ after τ, we denote the new sampling trajectory as $\tau' = \tau \circ v$, where $\circ$ represents concatenation.

Definition 2 (Successful Trajectory). *A sampling trajectory τ is* successful *if it represents an embedding. That is,*

$$|\tau| = |V_P| \quad and \quad \forall\, (u_i, u_j) \in E_P,\ (v_i, v_j) \in E_G.$$

Definition 3 (Number of Successful Extensions). *Given a sampling trajectory $\tau = (v_1, v_2, \ldots, v_k)$ with $0 \le |\tau| \le |V_P|$, the* number of successful extensions *n_τ is the total number of distinct successful trajectories that extend τ by adding vertices from V_G. Formally,*

$$n_\tau = \left| \left\{ \tau' \in {V_G}^{|V_P|} \;\middle|\; \begin{array}{l} \tau' = (v_1, v_2, \ldots, v_k, v_{k+1}, \ldots, v_{|V_P|}) \\ \tau' \text{ is a successful trajectory} \end{array} \right\} \right|.$$

For example, if $\tau = \emptyset$, the value n_τ is $C(G,P)$. Also for τ with $|\tau| = |V_P|$, n_τ is either 1 (τ is successful) or 0 (is not).

Theorem 1 (Ideal Sampling Distribution). *Suppose there exists at least one embedding of P in G. Assign a probability to each vertex $v \in V_G$ such that the probability is proportional to the number of successful extensions that can be generated when v is selected, i.e., $n_{\tau \circ v}$. Formally,*

$$\mathbf{f}_{ideal}(v \mid \tau) = \frac{n_{\tau \circ v}}{\sum_{x \in V_G} n_{\tau \circ x}}, \quad \mathbf{f}_{ideal}(\cdot \mid \tau) \in \mathbb{R}^{|V_G|}.$$

Then, a sampler using this probability assignment is ideal in that it returns $C(G,P)$ with zero variance.

Proof. Suppose we follow the above distribution $\mathbf{f}_{\text{ideal}}$ for sampling. Since τ is an empty sequence () at the beginning, we sample the first vertex v_1 from distribution $\mathbf{f}_1$, which is

$$\mathbf{f}_1(v) = \mathbf{f}_{\text{ideal}}(v \mid \tau = ()) = \frac{n_{(v)}}{\sum_{x \in V_G} n_{(x)}}.$$

We sample the first vertex v_1 with probability $\frac{n_{(v_1)}}{\sum_{x \in V_G} n_{(x)}}$. Then the next vertex is sampled from

$$\mathbf{f}_2(v) = \mathbf{f}_{\text{ideal}}(v \mid \tau = (v_1)) = \frac{n_{(v_1) \circ v}}{\sum_{x \in V_G} n_{(v_1) \circ x}}.$$

Repeating this process, at step i, we have the sampling trajectory $\tau = (v_1, v_2, \ldots, v_{i-1})$, and we sample the i-th vertex v_i from the distribution $\mathbf{f}_i$, which is:

$$\mathbf{f}_i(v) = \mathbf{f}_{\text{ideal}}(v \mid \tau = (v_1, \ldots, v_{i-1})) = \frac{n_{\tau \circ v}}{\sum_{x \in V_G} n_{\tau \circ x}}.$$

Since there exists at least one embedding of P in G, and we never sample a vertex v such that $n_{\tau \circ v} = 0$, the sampling process continues until $|\tau| = |V_P|$, and the sampling trajectory $\tau = (v_1, v_2, \ldots, v_{|V_P|})$ is always a successful one.

Moreover, for any such τ, the sampler output X_τ becomes

$$X_\tau = \frac{\sum_{x \in V_G} n_{(x)}}{n_{(v_1)}} \cdot \frac{\sum_{x \in V_G} n_{(v_1) \circ x}}{n_{(v_1, v_2)}} \cdot \ldots \cdot \frac{\sum_{x \in V_G} n_{(v_1, \ldots v_{|V_P|-1}) \circ x}}{n_{(v_1, \ldots v_{|V_P|})}} = C(G, P),$$

since $n_\tau = n_{(v_1, \ldots, v_{|V_P|})} = 1$, $\sum_{x \in V_G} n_{(x)} = C(G, P)$, and the terms cancel out as $\sum_{x \in V_G} n_{(v_1, \ldots v_i) \circ x} = n_{(v_1, \ldots v_i)}$. The sampler always returns the number of embeddings $C(G, P)$ with variance 0. □

4.2 Approximating the Ideal Distribution

We have established that sampling from the ideal distribution $\mathbf{f}_{\text{ideal}}(\cdot \mid \tau)$ yields an unbiased, zero-variance estimate. However, constructing the ideal probability distribution requires prior knowledge of the ratios of $n_{\tau \circ v}$, which necessitates knowing $C(G, P)$ beforehand. Since this is infeasible, we need an efficient method to build an approximate distribution $\mathbf{f}_{\text{approx}}(\cdot \mid \tau)$ that closely approximates $\mathbf{f}_{\text{ideal}}(\cdot \mid \tau)$ and yields unbiased results.

4.2.1 *Unbiasedness of a sampling distribution.*

Prior to constructing $\mathbf{f}_{\text{approx}}$, a natural question arises: can arbitrary sampling distributions be employed while still preserving the unbiasedness of the resulting estimator? The following theorem affirms this, under specific conditions.

Theorem 2. *Suppose the sampling distribution $\mathbf{f}(v \mid \tau)$ assigns a positive probability to every vertex $v \in V_G$ such that $n_{\tau \circ v} > 0$. That is, the distribution ensures that every successful trajectory is reachable. Then the sampler is an unbiased estimator of $C(G, P)$.*

Proof. Let X be the output of the sampler. By definition, the expectation of X is given by

$$\mathbb{E}[X] = \sum_\tau \mathbb{1}_{\{\tau \text{ is successful}\}} X_\tau \, p_\tau.$$

where X_τ represents the sampler's output for trajectory τ, and p_τ denotes the probability that the sampler follows trajectory τ. This expression holds because a sampler outputs $X_\tau = 0$ for any trajectory τ that is not successful.

First, note that each successful trajectory corresponds to a unique embedding of P. Second, every unique embedding is reachable through the sampling process. Therefore, for all $C(G, P)$ embeddings of G, there exist corresponding successful trajectories $\tau_1, \ldots, \tau_{C(G,P)}$. Thus we can write

$$\mathbb{E}[X] = \sum_\tau \mathbb{1}_{(\tau \text{ is successful})} X_\tau \, p_\tau = \sum_{i=1}^{C(G,P)} X_{\tau_i} p_{\tau_i}.$$

Also, for each successful τ_i, the sampler outputs $X_{\tau_i} = \frac{1}{p_{\tau_i}}$, which is well-defined. Therefore, the expected value becomes

$$\mathbb{E}[X] = \sum_{i=1}^{C(G,P)} X_{\tau_i} \, p_{\tau_i} = \sum_{i=1}^{C(G,P)} \frac{1}{p_{\tau_i}} \, p_{\tau_i} = \sum_{i=1}^{C(G,P)} 1 = C(G, P).$$

Thus, the sampler is an unbiased estimator of $C(G, P)$. □

4.2.2 *Generalized Approximation Framework.*

We begin by filtering out unpromising vertices by applying connectivity constraints and excluding already sampled vertices, following [7]. To sample the ith vertex (i.e. $i = |\tau| + 1$), we define the candidate set S_τ of promising vertices as

$$\mathrm{S}_\tau = \left\{ v \;\middle|\; v \notin \tau, \; v \in \bigcap_{u_j \in e \in \mathcal{B}, j < i} \mathcal{N}(v_j) \right\},$$

where $\mathcal{B}$ is the set of backward edges connecting u_i to already sampled vertices, that is

$$\mathcal{B} = \{\{u_j, u_i\} \in E_P \mid j < i\}.$$

We assign $\mathbf{f}_{\text{approx}}(v \mid \tau) = 0$ to all $v \notin \mathrm{S}_\tau$, because it is guaranteed that $n_{\tau \circ v} = 0$.

Our goal is now to approximate the ratios of $n_{\tau \circ v}$ for vertices $v \in \mathrm{S}_\tau$. The key idea is to consider the structural components of the pattern P in relation to the data graph G. We begin by grouping the vertices based on their distance from u_i within the subgraph $P_i = (V_i, E_i)$, which comprises the yet-to-be-sampled vertices of P:

$$V_i = \{u_j \in V_P \mid j \geq i\}, \; E_i = \{\{u_a, u_b\} \in E_P \mid u_a, u_b \in V_i\}. \quad (1)$$

We then define the k-hop vertex group as the set of vertices in P_i that are at a distance k from u_i in P_i. Using this information, we decompose $n_{\tau \circ v}$ into a product of terms, each accounting for different types of edges divided by k-hop information:

$$n_{\tau \circ v} \approx \prod_{k=1}^{D} T[\mathcal{F}_k](v) T[\mathcal{I}_k](v), \quad (2)$$

where

- $T[\mathcal{F}_k](v)$ is the term accounting for the set of *forward edges*, $\mathcal{F}_k$. Specifically, $\mathcal{F}_k \subseteq E_i$ consists of all edges that connect vertices at hop $(k-1)$ to vertices at hop k.
- $T[\mathcal{I}_k](v)$ is the term accounting for the set of *internal edges*, $\mathcal{I}_k$. Specifically, $\mathcal{I}_k \subseteq E_i$ consists of all edges among vertices at the same hop k.
- D is the maximum number of hops from u_i to any other connected vertex in P_i.

Note that Eq. (2) does not account for all edges of P. For example, there may be vertices in P_i that are not connected to u_i, or edges connecting vertices at hop k to previously sampled vertices. Although it is theoretically possible to consider all those edges, we find that focusing on edges of type $\mathcal{F}_k$ and $\mathcal{I}_k$ is sufficiently effective. Moreover, while calculating all D-hop terms can yield more accurate probability approximations, we find that it incurs excessive computational overhead with minimal improvements. We find that using the first three terms of the equation provides an effective approximation:

$$n_{\tau \circ v} \approx T[\mathcal{F}_1](v) \cdot T[\mathcal{I}_1](v) \cdot T[\mathcal{F}_2](v).$$

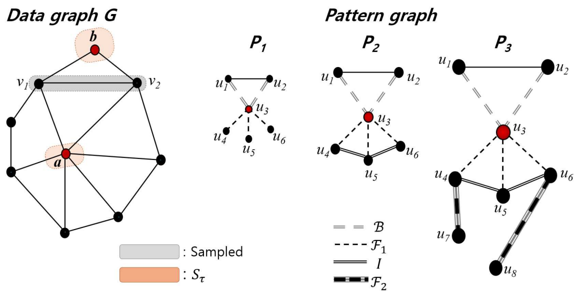


Figure 3: Example graph and patterns for the approximation.

This introduces a small degree of approximation error; however, such errors affect only the rate of convergence and do not impact the correctness of the embedding count $C(G, P)$. We will present, in Section 4.2.3, an equation for each term that ensures no vertex is assigned a zero probability unless it is certain, thereby guaranteeing unbiasedness (Theorem 2).

4.2.3 Approximation Details. In the example illustrated in Figure 3, we are sampling the third vertex $\pi(3) = u_3$ of the pattern P_1, P_2, P_3 from the graph G. Since u_3 is connected to u_1 and u_2, the corresponding vertex v_3 must be connected to both v_1 and v_2 in G. Therefore, the candidate set S_τ consists of the vertices adjacent to both v_1 and v_2 that are not already in τ; in this case, $\mathrm{S}_\tau = \{a, b\}$.

Now, consider the term $T[\mathcal{F}_1](v)$ with pattern P_1. In P_1, we have $\mathcal{B} = \{\{u_1, u_3\}, \{u_2, u_3\}\}$, and $\mathcal{F}_1 = \{\{u_3, u_4\}, \{u_3, u_5\}, \{u_3, u_6\}\}$. For a vertex $v \in \mathrm{S}_\tau$, it is already connected to the previously sampled vertices with $|\mathcal{B}|$ edges. Thus, we need to choose $|\mathcal{F}_1|$ edges out of $d(v) - |\mathcal{B}|$ possible neighbors. Thus we use,

$$T[\mathcal{F}_1](v) = \binom{d(v) - |\mathcal{B}|}{|\mathcal{F}_1|}.$$

For example, for a vertex $a \in \mathrm{S}_\tau$, $T[\mathcal{F}_1](a) = \binom{6-2}{3}$.

Next consider the term $T[\mathcal{I}_1]$ with a more general pattern P_2. In this case, $\mathcal{I}_1 = \{\{u_4, u_5\},\ \{u_5, u_6\}\}$. To account for these edges, we must estimate the probability p_v that $|\mathcal{I}_1|$ necessary edges exist among the $|\mathcal{F}_1|$ neighbors of v. One naive way to estimate p_v is by using the local clustering coefficient C_v of v [85]. Since C_v represents the probability that two neighbors of v are connected by an edge, and there are $|\mathcal{I}_1|$ such edges required, we can estimate p_v as $C_v^{|\mathcal{I}_1|}$.

However, naively estimating p_v as $C_v^{|\mathcal{I}_1|}$ tends to overemphasize the effect of C_v, leading to inaccurate approximations. This is because real-world graphs tend to form clusters [34, 35, 56, 57, 84], and that $|\mathcal{I}_1|$ edges are not independent from each other. For example, the probability that an edge $\{a, b\}$ exists given that $a, b, c \in \mathcal{N}(v)$ and edges $\{a, c\},\ \{b, c\}$ exist is generally higher than C_v due to the clustering effect.

Therefore, instead of multiplying C_v for every edge in $\mathcal{I}_1$, we construct an effective subset $\mathcal{I}_{\text{effective}} \subseteq \mathcal{I}_1$ of edges that can be considered independent. For this, we consider the subgraph induced by $\mathcal{I}_1$: $P_{\text{ind}} = (V_{\text{ind}}, E_{\text{ind}})$, where $V_{\text{ind}} = \{u \mid u \in e \in \mathcal{I}_1\}$ and $E_{\text{ind}} = \mathcal{I}_1$. We then select a spanning forest of P_{ind}, whose edge set is $\mathcal{I}_{\text{effective}}$. We then estimate

$$T[\mathcal{I}_1](v) = C_v^{|\mathcal{I}_{\text{effective}}|}.$$

In the case of pattern P_2, the induced subgraph P_{ind} forms a simple path over three vertices u_4, u_5, u_6. Since a path is already a tree, the entire edge set $\mathcal{I}_1$ serves as its own spanning tree. Therefore, for this particular case, we set $\mathcal{I}_{\text{effective}} = \mathcal{I}_1 = \{\{u_4, u_5\},\ \{u_5, u_6\}\}$. We then have $T[\mathcal{I}_1](a) = C_a^2 = (5/15)^2$.

Lastly, $T[\mathcal{F}_2](v)$ accounts for the two-hop topology of the pattern. For the pattern P_3, $\mathcal{F}_2 = \{\{u_4, u_7\}, \{u_6, u_8\}\}$. For each edge in $\mathcal{F}_2$, the number of successful extensions is estimated to increase proportionally to the degree of the corresponding vertices. For example, $T[\mathcal{F}_2](v) = d(v_4) \cdot d(v_6)$ appropriately accounts for potential connections to u_7 and u_8. However, since we do not know the exact 1-hop neighbors (v_4, v_6) that will be sampled in the future, we use the average degree of the neighbors of v for actual computation. Thus,

$$T[\mathcal{F}_2](v) = \left(\frac{\sum_{x \in \mathcal{N}(v)} d(x)}{d(v)} \right)^{|\mathcal{F}_2|}.$$

For example, $T[\mathcal{F}_2](a) = \left(\frac{4+4+3+3+3+3}{6}\right)^2$, considering neighbors of a starting from v_1 in a clockwise manner.

We provide a formal definition and an equation to calculate $\mathbf{f}_{\text{approx}}(\cdot \mid \tau)$.

Definition 4 (Auxiliary Arrays). *Given a pattern P with matching order $\pi : \{1, \ldots, |V_P|\} \rightarrow V_P$, so that $u_i = \pi(i) \in V_P$ is the i-th pattern vertex, we define the* Auxiliary Arrays A^π *as collections of arrays of sets:*

$$A^\pi = \left\{ \mathcal{B}^\pi,\ \mathcal{F}_1^\pi,\ \mathcal{F}_2^\pi,\ \mathcal{I}_{\text{effective}}^\pi \right\},$$

where each $\mathcal{B}^\pi, \mathcal{F}_1^\pi, \mathcal{F}_2^\pi, \mathcal{I}_{\text{effective}}^\pi$ is an array indexed by $i = 1, \ldots, |V_P|$, and each element $\mathcal{B}^\pi[i], \ldots, \mathcal{I}_{\text{effective}}^\pi[i]$ is the set associated with the vertex $u_i = \pi(i)$.

Based on A^π, the equation for approximating $n_{\tau \circ v}$, where v is the candidate for the i-th vertex v_i, is given by

$$n_{\tau \circ v} \approx \underbrace{\binom{d(v) - |\mathcal{B}^\pi[i]|}{|\mathcal{F}_1^\pi[i]|}}_{T[\mathcal{F}_1]} \times \underbrace{C_v^{|\mathcal{I}_{\text{effective}}^\pi[i]|}}_{T[\mathcal{I}_1]} \times \underbrace{\left(\frac{\sum_{x \in \mathcal{N}(v)} d(x)}{d(v)} \right)^{|\mathcal{F}_2^\pi[i]|}}_{T[\mathcal{F}_2]}. \quad (3)$$

We compute this value for all candidate vertices $v \in \mathrm{S}_\tau$, and use normalized values as the distribution $\mathbf{f}_{\text{approx}}(\cdot \mid \tau)$.

4.3 Analysis on Sample Complexity

To develop theoretical insight on our sampler proposed in Section 4.2.3, we analyze the sample complexity. For each successful trajectory τ_i, the sampler's output can be written as

$$X_{\tau_i} = C(G, P) \times (1 + \eta_i),$$

where η_i denotes the multiplicative error associated with trajectory τ_i arising from the use of $\mathbf{f}_{\text{approx}}$ in place of the ideal distribution $\mathbf{f}_{\text{ideal}}$. Since an unsuccessful trajectory yields $X_\tau = 0$, the second moment of the sampler output is

$$\mathbb{E}[X^2] = \sum_{\tau} \mathbb{1}_{\{\tau \text{ successful}\}} X_\tau^2 p_\tau = \sum_{i=1}^{C(G,P)} X_{\tau_i}$$
$$= C(G, P)^2 \, (1 + \overline{\eta}),$$

where $\overline{\eta}$ is the average multiplicative error

$$\overline{\eta} = \frac{1}{C(G,P)} \sum_{i=1}^{C(G,P)} \eta_i.$$

Combining Theorem 2 with Chebyshev's inequality, a $(1\pm\epsilon)$-relative error is achieved with probability at least $1-\delta$ whenever

$$N \geq \frac{\overline{\eta}}{\epsilon^2 \delta}. \tag{4}$$

This yields an explicit relationship between N (the number of samplers required), the accuracy of $\mathbf{f}_{\text{approx}}$, and the parameters (ϵ, δ). The closer $\mathbf{f}_{\text{approx}}$ is to $\mathbf{f}_{\text{ideal}}$, the closer $\overline{\eta}$ is to zero, and the fewer samplers are required.

To illustrate, consider the simplest non-degenerate pattern, the triangle. For a successful trajectory $\tau_i = (v_1, v_2, v_3)$, the multiplicative error η_i for the proposed sampler output is

$$\eta_i = \left[\frac{1}{d(v_1)\big(d(v_1)-1\big)C_{v_1}} \sum_{v \in \mathcal{N}(v_1)} \big(d(v)-1\big) \cdot \frac{|\mathcal{N}(v_1) \cap \mathcal{N}(v_2)|}{d(v_2)-1} \right] - 1. \tag{5}$$

The ratio $\frac{|\mathcal{N}(v_1)\cap\mathcal{N}(v_2)|}{d(v_2)-1}$ approximates the fraction of edges emanating from a neighbor of v_1 that remain inside $\mathcal{N}(v_1)$. Because $\sum_{v\in\mathcal{N}(v_1)}(d(v)-1)$ counts all edges incident to $\mathcal{N}(v_1)$, the product

$$\sum_{v \in \mathcal{N}(v_1)} \big(d(v)-1\big) \frac{|\mathcal{N}(v_1) \cap \mathcal{N}(v_2)|}{d(v_2)-1}$$

serves as a proxy for the number of edges internal to $\mathcal{N}(v_1)$, whose exact value is $d(v_1)\big(d(v_1)-1\big)C_{v_1}$. Consequently, regardless of the data graph G, Eq. (5), and thus $\overline{\eta}$, is typically close to zero.

On the other hand, for a sampler based on `NS-prune` without symmetry breaking, the error is given as

$$\eta'_i = \left[\frac{2|E_G| \cdot |\mathcal{N}(v_1) \cap \mathcal{N}(v_2)|}{C(G,P)} \right] - 1. \tag{6}$$

If G is uniform, every edge produces the same number of triangles and Eq. (6) is exactly 0. However, as the graph deviates from uniformity, η'_i deviates further from 0, and more samplers are required. Thus, Eqs. (4) to (6) suggest that the approximation strategy in Section 4.2.3 can be substantially more effective than existing approaches.

4.4 Preprocessing Data Graphs

For a given graph G, we preprocess three pieces of information, which are used for the calculation of $\mathbf{f}_{\text{approx}}$ from Eq. (3).

(1) The binomial coefficient table $\binom{n}{k}$ for $n \geq k$.
(2) The average neighbor degree information $\frac{\sum_{x\in\mathcal{N}(v)} d(x)}{d(v)}$.
(3) The clustering coefficient C_v.

While the first two items are relatively cheap to process during the graph loading phase, computing the exact value of C_v is time-consuming since it is equivalent to counting all the triangles in the graph. Instead, we estimate C_v by sampling. For each vertex v, we perform sampling $d(v)$ times and estimate C_v from the collected data. We will discuss the preprocessing time in Section 6. We also note that we bound the estimated clustering coefficient values to a small nonzero constant so that $\mathbf{f}_{\text{approx}}$ does not contain any unintended zeros which guarantees the assumption of Theorem 2.

5 AGIS SYSTEM DESIGN

Based on $\mathbf{f}_{\text{approx}}$, we build the AGIS system. We describe how to strategically use $\mathbf{f}_{\text{approx}}$ to accelerate the system (Section 5.1), how to build the matching order (Section 5.2), followed by the complete procedure (Section 5.3), and system overview with implementation details (Section 5.4).

5.1 Structure-Informed Neighbor Sampling

The proposed $\mathbf{f}_{\text{approx}}$ can dramatically reduce the required number of samplers for convergence. However, naively using $\mathbf{f}_{\text{approx}}$ can sometimes lead to longer execution time compared to using $\mathbf{f}_{\text{uniform}}$. While sampling from $\mathbf{f}_{\text{uniform}}$ can be done in $O(1)$ time, sampling from $\mathbf{f}_{\text{approx}}$ takes $O(|V_G|)$ time for constructing the distribution. Even though the number of samplers is still significantly smaller than that of using $\mathbf{f}_{\text{uniform}}$, the overall latency can be longer.

To address this problem, the proposed *structure-informed neighbor sampling* comprises a decision heuristic that determines the type of sampling distribution for each vertex based on the structure of the remaining unsampled subgraph of P. In the decision heuristic, we consider the density of the remaining unsampled subpattern. We find that as the unsampled portion of the pattern gets denser, $n_{\tau\circ v}$ becomes less predictable, and utilizing $\mathbf{f}_{\text{uniform}}$ is more beneficial.

Using the difference between the number of edges and the number of vertices to represent density, we define the following decision function:

Definition 5 (Decision Function). *Given a pattern P, a matching order $\pi : \{1,\ldots,|V_P|\} \longrightarrow V_P$, so that $u_i = \pi(i) \in V_P$, and a threshold $\beta \leq 1$, the decision function $\mathcal{D}^\pi : \{1,\ldots,|V_P|\} \longrightarrow \{0,1\}$ is defined as follows:*

(1) ***Modeling the Certainty of Using $\mathbf{f}_{approx}$:***
- *Define the set of vertices at least two hops away from u_i in $P_i = (V_i, E_i) \subseteq P$ (Eq. (1)):*
$$V_i^{\geq 2hop} = \{u \in V_i \mid 2 \leq dist_{P_i}(u_i, u) < \infty\}.$$
- *Define the set of edges connected to $V_i^{\geq 2hop}$:*
$$E_i^{\geq 2hop} = \left\{ \{u_a, u_b\} \in E_P \,\middle|\, u_a \in V_i^{\geq 2hop} \text{ or } u_b \in V_i^{\geq 2hop} \right\}.$$
- *Define the certainty to use $\mathbf{f}_{approx}$:*
$$Certainty(i) = 1 - \frac{|E_i^{\geq 2hop}| - |V_i^{\geq 2hop}|}{|V_i|}.$$
The value is bounded above by one, i.e., $Certainty(i) \leq 1$.

(2) ***Decision Function:***
$$\mathcal{D}^\pi(i) = \mathbb{1}_{(Certainty(i) \geq \beta)}$$

Here, we use $\beta = 0.8$ as the default value. $\mathcal{D}^\pi(i) = 1$ indicates that we use $\mathbf{f}_{\text{approx}}$ for sampling v_i, and $\mathcal{D}^\pi(i) = 0$ means we use $\mathbf{f}_{\text{uniform}}$. This allows us to combine the strengths of both distributions based on the topology of the remaining subpattern. Note that this can be precomputed only from P with negligible cost.

5.2 Matching Order Construction

Since the decision function $\mathcal{D}^\pi$ depends on the matching order π, it is crucial to construct π wisely to maximize the use of $\mathbf{f}_{\text{approx}}$. To

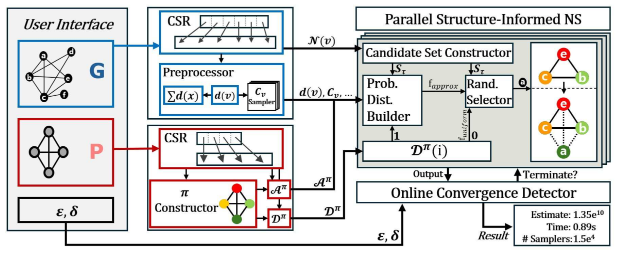


Figure 4: Overview of AGIS System.

achieve this, we propose an algorithm that incrementally builds the matching order in a greedy manner, as detailed in Algorithm 2.

Algorithm 2 Matching Order Construction

```
1:  Initialize empty list $\pi \leftarrow [\,]$
2:  for $i$ in $[1, |V_P|]$ do
3:      if $i = 1$ then
4:          $\mathcal{N} \leftarrow V_P$
5:      else
6:          $\mathcal{N} \leftarrow \{u \in V_P \setminus \pi \mid \exists u' \in \pi$ such that $\{u, u'\} \in E_P\}$
7:      end if
8:      for $u \in \mathcal{N}$ do
9:          $\pi_u \leftarrow \pi + u,\ s(u) \leftarrow |\mathcal{F}_1^{\pi_u}[i]| + |\mathcal{I}_1^{\pi_u}[i]|$
10:     end for
11:     $\pi \leftarrow \pi + \mathrm{argmax}_{u \in \mathcal{N}}\, s(u)$
12: end for
13: return $\pi$
```

As shown in Lines 3–7, we first construct a candidate set $\mathcal{N}$ of neighboring vertices so that we always explore the pattern in a connected fashion. In Lines 8–10, we calculate the score $s(u)$ for all $u \in \mathcal{N}$, where $|\mathcal{F}_1^{\pi_u}[i]|$ and $|\mathcal{I}_1^{\pi_u}[i]|$ are defined as in Section 4.2.3, assuming that vertex $u \in V_P$ is selected as the next vertex in the matching order. In Line 11, by selecting the vertex that maximizes this sum, we prioritize vertices with the largest unsampled 1-hop structures. This approach results in $\mathbf{f}_{\text{approx}}$ being selected more frequently, as the *Certainty* value in Definition 5 will be higher.

5.3 Complete Sampling Procedure

The procedure of the sampling algorithm used in AGIS is displayed in Algorithm 3. In Lines 4–9, we construct the candidate set S_τ using connectivity constraints. If no possible vertices exist (Lines 10), we return a value of 0 and terminate the sampling process. Otherwise, we build a probability distribution function according to the decision function $\mathcal{D}^\pi$ (Lines 11–17). We then sample one vertex from S_τ following this distribution and update the probability p accordingly (Lines 18–19). By repeating this vertex sampling process $|V_P|$ times, we return $1/p$ as the sampler output.

We note that sampling solely based on $\mathbf{f}_{\text{uniform}}$ is equivalent to ScaleGPM's `NS-prune` procedure without applying symmetry breaking. The definition of $\mathbf{f}_{\text{uniform}}$ varies depending on step i. For $i = 1$ (sampling the first vertex), $\mathbf{f}_{\text{uniform}}$ is *edge-uniform*, meaning that vertices are sampled with probability proportional to their degrees. For $i \geq 2$, $\mathbf{f}_{\text{uniform}}$ is *vertex-uniform*, so each vertex in S_τ is sampled with equal probability.

Algorithm 3 Structure-Informed Neighbor Sampling

```
1:  procedure SAMPLE($G, P, \pi, \mathcal{D}^\pi, \mathcal{A}^\pi$)
2:      Initialize $\tau \leftarrow [\,], p \leftarrow 1.0$
3:      for $i$ in $[1, |V_P|]$ do
4:          ▷ Build set $S_\tau$ of candidate vertices
5:          if $i = 1$ then
6:              $S_\tau \leftarrow V_G$
7:          else
8:              $S_\tau \leftarrow \{v \mid v \notin \tau, v \in \bigcap_{\{u_j,u_i\} \in E_P, j<i} \mathcal{N}(v_j)\}$
9:          end if
10:         if $|S_\tau| = 0$ then return 0
11:         ▷ Build probability distribution
12:         if $\mathcal{D}^\pi(i) = 1$ then
13:             calculate $\mathbf{f}_{\text{approx}}$ using $(S_\tau, \mathcal{A}^\pi, i)$ with Eq. (3)
14:             $\mathbf{f} \leftarrow \mathbf{f}_{\text{approx}}$
15:         else
16:             $\mathbf{f} \leftarrow \mathbf{f}_{\text{uniform}}$
17:         end if
18:         $v \leftarrow rand\_select(S_\tau, \mathbf{f})$
19:         $\tau \leftarrow \tau + v, p = p \cdot \mathbf{f}(v)$
20:     end for
21:     return $\frac{1}{p}$
22: end procedure
```

5.4 System Overview

Based on structure-informed neighbor sampling, we propose AGIS, a system for fast approximate graph pattern mining. The high-level sketch of AGIS is given in Figure 4. It first receives four inputs from the user: (1) a graph file G, (2) a pattern file P, (3) ϵ, and (4) δ. The graph is read and loaded in CSR format. If the graph has not been preprocessed, the preprocessor calculates additional information (Section 4.4). For the pattern, the system first constructs the matching order π which is then used to calculate $\mathcal{D}^\pi$ and $\mathcal{A}^\pi$.

Once the information is ready, the sampling engine is run in parallel, where each thread is assigned an individual sampler. For every fixed number of iterations, the sampler outputs are checked for convergence by the online convergence detector (Algorithm 1) proposed by ScaleGPM , utilizing ϵ and δ. Finally, the results, such as the estimated pattern count, are output to the user.

When the queried pattern P is a clique, a popular approach is to apply orientation optimization [24, 40, 73, 79]. It converts the undirected graph into a directed acyclic graph (DAG) by removing edges pointing from high-degree vertex to low-degree vertex. Similar to ScaleGPM [7], which utilizes this method, we apply the same optimization for clique patterns.

6 EVALUATION

AGIS is implemented in C++ with OpenMP for multithreading. We evaluated AGIS over a diverse set of data graphs and patterns. We used real-world graphs spanning a wide range of sizes, as summarized in Table 1. For the patterns to be mined, we selected a diverse set of patterns, partly drawn from prior works [7, 91], as

Graph	$\lvert V_G \rvert$	$\lvert E_G \rvert$	Max Degree
LiveJournal (Lj) [9, 48]	4.8×10^6	4.3×10^7	2.0×10^4
Uk-2002 (Uk) [15–17]	1.8×10^7	2.6×10^8	1.9×10^5
Twitter (Tw) [47, 48]	4.2×10^7	1.2×10^9	3.0×10^6
Friendster (Fs) [48, 88]	6.6×10^7	1.8×10^9	5.2×10^3
Gsh-2015 (Gsh) [16, 17]	9.9×10^8	2.6×10^{10}	5.9×10^7

Table 1: Graph datasets used for evaluation.

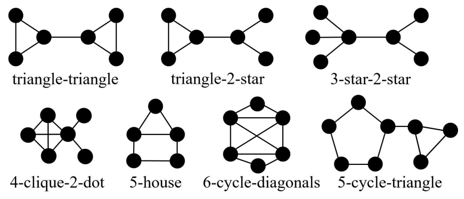


Figure 5: Patterns used for evaluation.

Instance	4-clique				4-clique-2-dot				6-clique			
	Lj	Uk	Tw	Fs	Lj	Uk	Tw	Fs	Lj	Uk	Tw	Fs
AGIS	**0.002**	**0.006**	**0.016**	0.028	**0.003**	**0.069**	**5.330**	**0.045**	**0.001**	**0.005**	**0.017**	**0.143**
ScaleGPM	0.007	0.014	0.018	**0.005**	0.052	191.9	1930	0.089	0.095	0.843	0.377	2.460
Arya	504.4	688.6	TO	TO	2126	1528	TO	TO	TO	TO	TO	TO
ASAP	958.2	462.4	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO
Peregrine	6.779	128.0	TO	199.5	TO	TO	TO	TO	TO	TO	TO	1114
Instance	**3-star-2-star**				**triangle-2-star**				**triangle-triangle**			
	Lj	Uk	Tw	Fs	Lj	Uk	Tw	Fs	Lj	Uk	Tw	Fs
AGIS	**0.002**	**0.024**	**0.840**	**0.028**	**0.002**	**0.081**	**4.701**	**0.030**	**0.001**	**0.013**	**4.738**	**0.044**
ScaleGPM	404.1	TO	TO	0.054	0.197	40.53	5009	0.043	0.025	0.186	33.75	0.096
Arya	216.4	1345	154.1	135.4	140.2	177.2	632.1	1640	686.3	1827	9148	TO
ASAP	2372	TO	TO	3.067	TO	TO	TO	6273	4628	TO	TO	TO
Peregrine	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO	TO

Table 2: Execution time (sec.) for patterns where $\mathcal{D}^{\pi} = 1$. TO: timed out. Fastest time in bold.

illustrated in Figure 5. We compared AGIS against the state-of-the-art AGPM systems ASAP [43], Arya [91] and ScaleGPM [7] and an exact mining system Peregrine [44]. We used the official implementations for Peregrine, and Arya. We faithfully reproduced ASAP from scratch as their codes are not publicly available. Because the official ScaleGPM implementation requires custom implementation for individual patterns, we implemented a general-purpose version of ScaleGPM and, where applicable, reported the minimum execution time observed between our implementation and the official code.

For all systems, we excluded the time required to load the input graphs. We also excluded the time spent on pattern-specific preprocessing steps such as solving linear programs for pattern decomposition in Arya, constructing automorphisms and restriction sets for ScaleGPM, and computing $\mathcal{A}^{\pi}$, $\mathcal{D}^{\pi}$ in AGIS. For ScaleGPM, we used only the strict mode (default) of ScaleGPM. This is because the loose mode of ScaleGPM uses graph sparsification with exact counting, which does not use online convergence and thus cannot provide results with high confidence.

To validate the estimated values, we compared the results against Peregrine for all (graph, pattern) pairs that Peregrine can mine within a moderate time. For pairs where Peregrine fails to produce an exact count, the ground truth is unknown. However, for all of the cases, we confirmed that ScaleGPM and AGIS produce similar estimates, ensuring that $\mid (X_s - X_a)/((X_s + X_a)/2) \mid < 2\epsilon$, where X_s and X_a are the results from ScaleGPM and AGIS, respectively.

For all baselines and AGIS, the experiments were conducted on an AMD Ryzen Threadripper PRO 7985WX with 48 physical cores with 512GB DRAM. We used a timeout of 1.0×10^4 s for all runs.

6.1 Overall Performance

Table 2 and Table 3 show the main results for diverse patterns. We follow the most common practice of $\epsilon = 0.1, \delta = 0.01$ if not specified otherwise. Overall, AGIS achieves the fastest speed for the majority of patterns and successfully mines all patterns without encountering timeouts. Specifically, AGIS obtains 28.5 × speedup (using 10^4s for timeouts) against ScaleGPM, and 63,108× speedup against Peregrine in geometric mean.

As expected, Peregrine, the exact GPM system fails to produce results within the given time limit for most cases. For graphs larger than Lj, Peregrine can only mine clique patterns where heavy pruning can be applied. This clearly demonstrates the need for AGPM systems over exact mining systems.

Among the baselines, ScaleGPM achieves the best overall performance. We also observe that ScaleGPM performs the best for clique patterns due to the benefits of DAG orientation optimization. However, ScaleGPM requires a long time to converge for some (graph pattern) pairs. For instance, a 3-star-2-star pattern on Lj takes more than 100,000× longer to converge compared to AGIS. This observation aligns with those in Section 3.1, where patterns containing bridges pose significant challenges under skewed degree distributions.

We can further observe this phenomenon by comparing results on the Fs graph. Fs is distinctive due to its near-uniform degree distribution. As shown in Table 1, Fs contains more edges than Tw, yet its maximum degree is only about 0.2% of that of Tw. Since

Instance	5-house		6-cycle-diagonals		5-cycle-triangle	
	`Lj`	`Tw`	`Lj`	`Tw`	`Lj`	`Tw`
AGIS	**0.003**	**2.775**	**0.003**	**0.580**	**0.036**	**188.9**
ScaleGPM	0.009	31.72	0.022	27.25	0.041	2002
Arya	65.83	5490	TO	TO	TO	TO
ASAP	117.8	TO	TO	TO	TO	TO
Peregrine	669.1	TO	1483	TO	TO	TO

Table 3: Execution time (sec.) for patterns where $\mathcal{D}^\pi \neq 1$. TO: timed out.

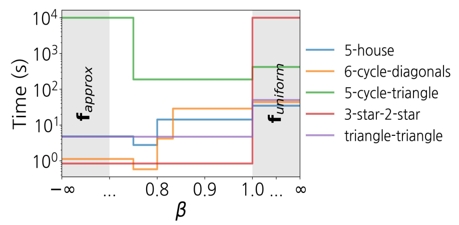


Figure 6: Impact of β on execution time over `Tw` graph.

Pattern	4-clique	5-house	triangle-triangle
AGIS	0.443	**89.11**	**1120**
ScaleGPM	**0.130**	731.1	TO
Arya	OoM	OoM	OoM
ASAP	TO	TO	TO
Peregrine	TO	TO	TO

(a) Execution time on `Gsh` graph.

	`Lj`, 4-clique			`Lj`, 5-house		
Error bound ϵ	10%	1%	0.1%	10%	1%	0.1%
AGIS	**0.002**	**0.008**	**0.528**	**0.003**	**0.026**	**2.314**
ScaleGPM	0.008	0.689	68.86	0.012	1.240	129.0
Arya	504.4	8545	TO	65.83	1037	3910
ASAP	958.2	1901	TO	117.8	573.6	TO

(b) Execution time for different error bounds.

Table 4: Execution time (sec.) for (a) huge graph and (b) different error bounds. TO: timed out. Fastest time in bold.

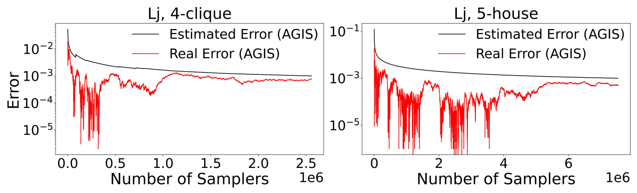


Figure 7: Real error compared to the estimated error from online convergence algorithm on AGIS.

the degree distribution in `Fs` is not highly skewed, ScaleGPM effectively handles 3-star-2-star and triangle-2-star patterns. Additionally, from a more fundamental standpoint, $\mathbf{f}_{\text{uniform}}$ takes an unintended advantage on `Fs`, since $\mathbf{f}_{\text{uniform}}$ will be closer to $\mathbf{f}_{\text{ideal}}$.

Furthermore, we can understand why Arya outperforms ScaleGPM on such patterns for highly skewed graphs through a similar reasoning process. This is because, unlike ScaleGPM, Arya attempts to achieve convergence on a subsampled graph, thereby effectively reducing the range of its degree distribution. However, as noted in Section 2.3, ELP provides no convergence guarantees and the results of the ELP heuristic are observed to be highly unstable.

The patterns are grouped into two tables. For the patterns examined in Table 2, the certainty values (Definition 5) are always maximal (equal to 1). Thus, regardless of which $\beta \leq 1$ value is chosen, AGIS consistently uses $\mathbf{f}_{\text{approx}}$. In contrast, for the patterns in Table 3, the certainty values are not consistently equal to 1. Specifically, in the 5-house pattern one of five vertices is sampled uniformly; in the 6-cycle-diagonals one of six; and in the 5-cycle-triangle three of eight. By testing for numerous patterns including those in Figure 6, we find that selecting a threshold value of $\frac{4}{5} \leq \beta \leq \frac{5}{6}$ yields robust and strong performance across a variety of patterns. Figure 6 illustrates how β affects the performance. Here, $\beta = -\infty$ indicates that $\mathbf{f}_{\text{approx}}$ is always used, whereas $\beta = 1$ denotes the most conservative use of $\mathbf{f}_{\text{approx}}$. For experimental purposes, we extend the range of β to include values where $\beta > 1$, which is equivalent to always choosing $\mathbf{f}_{\text{uniform}}$. As observed, employing $\mathbf{f}_{\text{approx}}$ ($\beta = -\infty$) generally results in faster convergence compared to using $\mathbf{f}_{\text{uniform}}$ ($\beta = +\infty$). Nevertheless, every pattern achieves its optimal performance for some value of β within the range $-\infty < \beta \leq 1$, indicating that an optimal point exists between the extremes of exclusively utilizing $\mathbf{f}_{\text{approx}}$ and exclusively utilizing $\mathbf{f}_{\text{uniform}}$. By utilizing the proposed $\mathcal{D}^\pi$, AGIS effectively balances the use of $\mathbf{f}_{\text{approx}}$ and $\mathbf{f}_{\text{uniform}}$.

Table 4a presents the performance results on the massive graph `Gsh`, containing more than 25 billion edges. As shown, only AGIS successfully scales to this large size. ScaleGPM is faster solely for the 4-clique pattern, where the overhead of building $\mathbf{f}_{\text{approx}}$ outweighs the complexity of the pattern. In addition, Table 4b reports the performance under different error bounds. As indicated, AGIS is the only framework that can handle $\epsilon = 10^{-3}$ within a few seconds. We also verified that the returned counts respect the given ϵ. In Figure 7, we plot the estimated error provided by the online convergence detector against the actual error, measured using exact mining results. For error levels up to 10^{-3}, the real error remains bounded by the estimated error, confirming that the online convergence method functions effectively within AGIS.

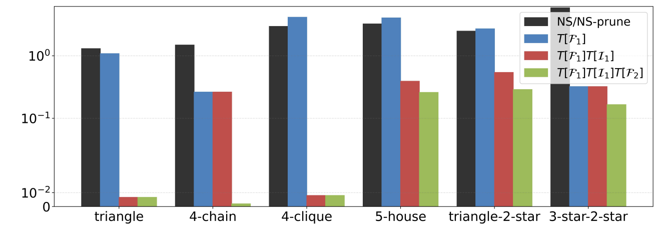
We note that there are a few corner cases where AGIS is slower than ScaleGPM (4-clique on the `Fs`, `Gsh`). The upfront cost of computing the full-graph vertex sampling distribution can outweigh its benefits when the target pattern is small and the ideal distribution is nearly uniform, rendering the sophisticated setup unnecessary.

6.2 Analysis on the Approximate Distribution

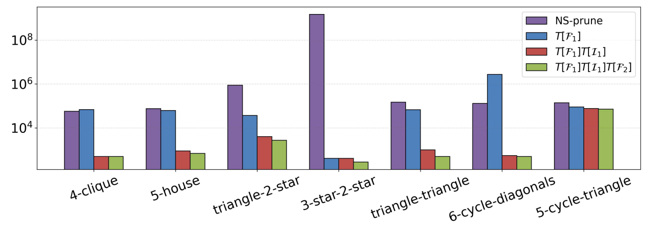
We now examine our estimation technique for the sampling distribution in greater detail. Table 5 compares the number of samplers required for convergence between AGIS and ScaleGPM, both of which employ the same convergence detection algorithm. As shown, AGIS requires orders of magnitude fewer samplers. For instance, ScaleGPM demands three million times more samplers to achieve

Instance	4-clique		4-cliqe-2-dot		6-clique		3-star-2-star		triangle-2-star		triangle-triangle	
	Lj	Tw	Lj	Tw	Lj	Tw	Lj	Tw	Lj	Tw	Lj	Tw
AGIS (Structure-Informed NS)	**5.0e2**	**2.3e3**	**5.0e2**	**4.0e3**	**7.5e2**	**1.0e3**	**5.0e2**	**5.0e2**	**2.8e3**	**1.4e4**	**5.0e2**	**1.1e4**
ScaleGPM (NS-prune)	5.7e4	1.3e5	2.9e5	8.2e7	1.2e6	2.8e6	1.5e9	X	8.8e5	5.6e7	1.5e5	2.3e6

Table 5: Number of samplers needed for convergence. X for failure in convergence. The smallest number is in bold.



(a) KL divergence from each distribution ($\mathbf{f}_{\text{approx}}$ and $\mathbf{f}_{\text{uniform}}$) to the ideal distribution $\mathbf{f}_{\text{ideal}}$.



(b) Number of samplers needed for convergence.

Figure 8: Efficiency of the proposed approximation technique. Tested on Lj graph.

convergence for the 3-star-2-star pattern in Lj. This highlights that the fundamental speedup of AGIS arises from leveraging a well-calculated $\mathbf{f}_{\text{approx}}$ rather than relying on a naive $\mathbf{f}_{\text{uniform}}$.

Figure 8 demonstrates how $\mathbf{f}_{\text{approx}}$ improves as we incorporate additional terms. In Figure 8a, we first count the actual value of $n_{(v)}$ for every vertex $v \in V_G$. With this information, we calculate the ideal distribution for sampling the first vertex of the pattern, $\mathbf{f}_{ideal}(v|())$. We then measure the KL divergence of $\mathbf{f}_{\text{ideal}}$ for several other distributions: $\mathbf{f}_{\text{uniform}}$ of NS/NS-prune, and $\mathbf{f}_{\text{approx}}$ of AGIS. We analyze three incremental versions of $\mathbf{f}_{\text{approx}}$: (1) using only $T[\mathcal{F}_1]$, (2) adding $T[\mathcal{I}_1]$, and (3) further incorporating $T[\mathcal{F}_2]$, the full version employed in AGIS. This breakdown allows us to observe the effect of each added term on the accuracy of the approximation.

For all patterns, $\mathbf{f}_{\text{approx}}$ of AGIS achieves orders of magnitude smaller KL divergence than $\mathbf{f}_{\text{uniform}}$, indicating a much closer approximation to $\mathbf{f}_{\text{ideal}}$. We also find that all three terms are necessary. For instance, in 4-clique, using only the first term yields a KL divergence slightly larger than $\mathbf{f}_{\text{uniform}}$. By adding the term $T[\mathcal{I}_1]$, we decrease the value to nearly 0. Similarly, for 4-chain, including the final term $T[\mathcal{F}_2]$ further boosts the deduction in KL divergence.

Figure 8b presents the number of samplers required for convergence with incremental addition of the terms. Here, we include larger patterns for which it is not feasible to compute $\mathbf{f}_{\text{ideal}}$. As shown, having all three terms yields clear benefits. However, as discussed in Section 4.2, the marginal improvement from adding the third term is the smallest. This observation aligns with the notion that incorporating higher-order terms leads to diminishing returns, since edges further along the pattern introduce greater uncertainty.

6.3 Comparison with MCMC based method

Markov Chain Monte Carlo (MCMC)–based counting techniques [13, 26, 37, 83] provide an alternative to NS-based sampling methods for estimating the concentration, the proportion of a specific pattern among all patterns of the same size in a given graph. These techniques construct a higher-order auxiliary graph whose

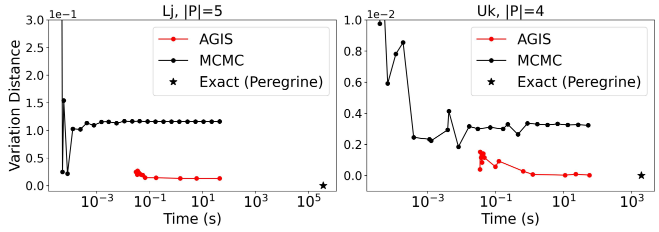
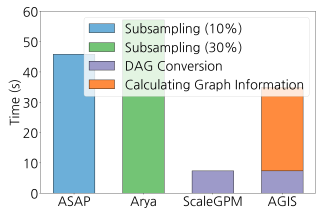


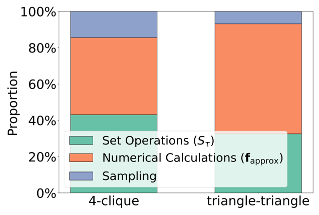
Figure 9: Variation distance to the true concentration.

vertices correspond to embeddings in the original graph. Two vertices of the auxiliary graph are adjacent if their embeddings differ by a small local modification, such as replacing, deleting, or adding a vertex. An MCMC algorithm then performs a random walk on this auxiliary graph until mixing, using the visitation frequencies to approximate the concentration.

Figure 9 compares AGIS with an MCMC-based counting algorithm on size-five patterns over the Lj graph and size-four patterns over the Uk graph. We implemented SRW2, the MCMC-based algorithm described in [26], which returns the estimated concentration for a specified pattern size. For AGIS, there are a total of 21 connected non-isomorphic size-five patterns and 6 connected non-isomorphic size-four patterns. We enumerated each pattern individually to obtain its approximate count, and then normalized these counts by the total number of patterns of the corresponding size to derive their concentrations. Ground-truth concentrations were obtained with the exact system Peregrine. Accuracy was measured by variation distance, defined as one half of the L_1 distance between concentration vectors. The figure shows that, while MCMC methods enjoy strong theoretical guarantees, AGIS produces much more accurate estimates within the same time budget. Specifically, at 1 s, the MCMC method exhibits variation distances that are 8.3× and 41.7× larger than those of AGIS.



(a) Tw preprocessing time.



(b) Tw mining time breakdown.

Figure 10: Execution time analysis.

Graph	Lj	Uk	Tw	Fs
Total Time (s)	0.477	3.069	34.93	55.37

Table 6: Preprocessing time across different graph datasets.

6.4 Execution Time Breakdown

We now consider the time spent on preprocessing. For ASAP and Arya, a subsampling step must be performed once per given graph. This step involves randomly sampling edges at a specified ratio, converting them into an undirected graph representation, and then refining them into a CSR format. For ScaleGPM, a directed acyclic graph (DAG) must be constructed for orientation optimization. In the case of AGIS, in addition to the DAG construction, we must also compute the required graph information (Section 4.4).

Figure 10a presents the preprocessing times for the four AGPM systems. By employing the sampling technique described in Section 4.4, the preprocessing phase for AGIS is not prohibitively slow, and remains comparable to that of ELP-based methods. For completeness, Table 6 reports the total preprocessing time for each graph. This one-time overhead is amortized over all pattern queries and thus negligible in practice, particularly when mining complex patterns or when mining under tight error bounds.

Lastly, we present a detailed timing breakdown of AGIS's runtime in Figure 10b. We partition the execution into three components: (1) set operations for constructing the candidate set $\mathbf{S}_\tau$, (2) numerical calculations for $\mathbf{f}_{\text{approx}}$ and (3) sampling via building a cumulative distribution. The analysis indicates that the computation of $\mathbf{f}_{\text{approx}}$ accounts for slightly more than half of the total runtime. However, as demonstrated in Section 6.1 and Section 6.2, this additional cost is justified, as a meticulously constructed $\mathbf{f}_{\text{approx}}$ leads to a substantial improvement in overall runtime.

7 RELATED WORK

Exact Graph Pattern Mining Systems. There have been numerous attempts to develop efficient exact Graph Pattern Mining systems. Systems such as [22, 25, 44, 50, 82], use various optimization techniques to improve performance. For instance, Dwarves-Graph [22] uses a decomposition algorithm and computes the counts for each decomposed pattern. GraphZero [50] uses a relabeling technique to generalize the orientation algorithm to arbitrary patterns. Peregrine [44] adopts pattern-aware algorithms to prune early, thus bypassing expensive isomorphism and canonicality checks. [23, 60] develops a combinatorial framework for efficient counting. Notably, [51, 71] employ a performance model with similar approximation logic to AGIS. They approximate the size of set intersections using a global constant (e.g., the total number of triangles), whereas AGIS leverages local fine grained information.

Other Approximate Counting Schemes. Beyond NS-based methods and MCMC-based methods, several alternative approaches have been developed for approximate subgraph counting. One example is the Color Coding (CC) approach [4, 5, 20, 75, 90]. The core idea is to assign each vertex a random color and focus on colorful matches, where all vertices in the pattern have distinct colors. This counting step can be done efficiently via dynamic programming, and by repeating the color assignment and counting procedure multiple times, the approximation can be refined.

Another line of research focuses on sparsifying the graph before applying exact counting procedures. Various studies have introduced distinct sparsification techniques to approximate specific patterns, including triangles [58, 78], cliques [69], 5-cycles [70], and butterfly patterns [67]. In a similar vein, ScaleGPM [7] introduces a "loose mode" that leverages sparsification when its primary strategy is anticipated to fail or become excessively time-consuming.

Some streaming algorithms [2, 81] maintain an edge reservoir by assigning each edge an inclusion probability proportional to its marginal contribution to the overall pattern count. This is analogous to AGIS, in which optimal sampling probabilities are proportional to the number of pattern instances incident on each vertex.

Counting Across Different Graph Types. While homogeneous graphs were considered in this paper, counting on heterogeneous graphs and hypergraphs is another important line of research. Type-aware counting techniques have been developed to efficiently handle constraints on heterogeneous graphs. These include combinatorial methods that exploit algebraic relationships to avoid exhaustive enumeration [55, 61, 63, 68, 74], discriminative mining approaches that discover frequent typed patterns [14, 21], and sampling-based estimators [64, 72].

For hypergraphs and simplicial complexes, recent systems [76, 89] leverage hyperedge features and parallelism. A growing body of work proposes sketch-based approximations for counting small sub-hypergraphs at scale [18, 46, 54, 80]. Recent studies adapt random-walk–based sampling [12, 41, 42, 62, 66].

8 CONCLUSION

In this paper, we introduced AGIS, a fast approximate graph pattern mining system that leverages a novel structure-informed neighbor sampling technique. By integrating structural properties from both data and pattern graphs, AGIS constructs an approximate ideal sampling distribution and then adaptively decides when to apply it for optimal efficiency. Experiments on diverse datasets show that AGIS outperforms state-of-the-art baselines by more than an order of magnitude (28.5× in geometric mean), achieving rapid convergence and scaling to graphs with tens of billions of edges.

ACKNOWLEDGMENTS

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) (RS-2024-00395134, RS-2024-00347394, RS-2023-00256081, RS-2021II211343, RS-2025-00564840). Jinho Lee is the corresponding author.

REFERENCES

[1] Charu C Aggarwal, Haixun Wang, et al. 2010. *Managing and mining graph data*. Vol. 40. Springer.

[2] Nesreen K Ahmed, Nick Duffield, Theodore L Willke, and Ryan A Rossi. 2017. On Sampling from Massive Graph Streams. *Proceedings of the VLDB Endowment* 10, 11 (2017).

[3] Leman Akoglu, Hanghang Tong, and Danai Koutra. 2015. Graph based anomaly detection and description: a survey. *Data mining and knowledge discovery* 29 (2015), 626–688.

[4] Noga Alon, Phuong Dao, Iman Hajirasouliha, Fereydoun Hormozdiari, and S Cenk Sahinalp. 2008. Biomolecular network motif counting and discovery by color coding. *Bioinformatics* 24, 13 (2008), i241–i249.

[5] Noga Alon, Raphael Yuster, and Uri Zwick. 1995. Color-coding. *J. ACM* 42, 4 (1995), 844–856.

[6] Naomi A Arnold, Peijie Zhong, Cheick Tidiane Ba, Ben Steer, Raul Mondragon, Felix Cuadrado, Renaud Lambiotte, and Richard G Clegg. 2024. Insights and caveats from mining local and global temporal motifs in cryptocurrency transaction networks. *Scientific Reports* 14, 1 (2024), 26569.

[7] Anna Arpaci-Dusseau, Zixiang Zhou, and Xuhao Chen. 2025. Accurate and Fast Approximate Graph Pattern Mining at Scale. *Proceedings of the VLDB Endowment* 18, 2 (2025), 93–107.

[8] Sepehr Assadi, Michael Kapralov, and Sanjeev Khanna. 2018. A simple sublinear-time algorithm for counting arbitrary subgraphs via edge sampling. *arXiv preprint arXiv:1811.07780* (2018).

[9] Lars Backstrom, Dan Huttenlocher, Jon Kleinberg, and Xiangyang Lan. 2006. Group formation in large social networks: membership, growth, and evolution. In *KDD*.

[10] Albert-László Barabási and Réka Albert. 1999. Emergence of scaling in random networks. *science* 286, 5439 (1999), 509–512.

[11] Albert-László Barabási, Réka Albert, and Hawoong Jeong. 1999. Mean-field theory for scale-free random networks. *Physica A: Statistical Mechanics and its Applications* 272, 1-2 (1999), 173–187.

[12] Hamid Beigy, Mohammad Mahini, Salman Qadami, and Morteza Saghafian. 2024. Approximating Simplet Frequency Distribution for Simplicial Complexes. arXiv:2402.16777 [cs.CG] https://arxiv.org/abs/2402.16777

[13] Mansurul A Bhuiyan, Mahmudur Rahman, Mahmuda Rahman, and Mohammad Al Hasan. 2012. Guise: Uniform sampling of graphlets for large graph analysis. In *ICDM*.

[14] Zhichun Li Zhenyu Wu Zhiyun Qian Xifeng Yan Ambuj K. Singh Guofei Jiang Bo Zong, Xusheng Xiao. 2015. Behavior query discovery in system-generated temporal graphs. *Proceedings of the VLDB Endowment* 9, 14 (2015), 240–251. https://doi.org/10.14778/2856318.2856320

[15] Paolo Boldi, Bruno Codenotti, Massimo Santini, and Sebastiano Vigna. 2004. Ubicrawler: A scalable fully distributed web crawler. *Software: Practice and Experience* 34, 8 (2004), 711–726.

[16] Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. 2011. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In *WWW*.

[17] Paolo Boldi and Sebastiano Vigna. 2004. The webgraph framework I: compression techniques. In *WWW*.

[18] Marco Bressan, Julian Brinkmann, Holger Dell, Marc Roth, and Philip Wellnitz. 2025. The Complexity of Counting Small Sub-Hypergraphs. arXiv:2506.14081 [cs.CC] https://arxiv.org/abs/2506.14081

[19] Marco Bressan, Flavio Chierichetti, Ravi Kumar, Stefano Leucci, and Alessandro Panconesi. 2018. Motif counting beyond five nodes. *Transactions on Knowledge Discovery from Data* 12, 4 (2018), 1–25.

[20] Venkatesan T Chakaravarthy, Michael Kapralov, Prakash Murali, Fabrizio Petrini, Xinyu Que, Yogish Sabharwal, and Baruch Schieber. 2016. Subgraph counting: Color coding beyond trees. In *IPDPS*.

[21] Yuan Fang; Wenqing Lin; Vincent W. Zheng; Min Wu; Jiaqi Shi; Kevin Chen-Chuan Chang and Xiao-Li Li. 2021. Metagraph-Based Learning on Heterogeneous Graphs. *IEEE Transactions on Knowledge and Data Engineering* 33, 1 (2021), 154–168.

[22] Jingji Chen and Xuehai Qian. 2020. Dwarvesgraph: a high-performance graph mining system with pattern decomposition. *arXiv preprint arXiv:2008.09682* (2020).

[23] Jingji Chen and Xuehai Qian. 2022. Decomine: A compilation-based graph pattern mining system with pattern decomposition. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*. 47–61.

[24] Xuhao Chen et al. 2022. Efficient and scalable graph pattern mining on GPUs. In *OSDI*.

[25] Xuhao Chen, Roshan Dathathri, Gurbinder Gill, Loc Hoang, and Keshav Pingali. 2021. Sandslash: a two-level framework for efficient graph pattern mining. In *ICS*.

[26] Xiaowei Chen, Yongkun Li, Pinghui Wang, and John Lui. 2016. A general framework for estimating graphlet statistics via random walk. *arXiv preprint arXiv:1603.07504* (2016).

[27] Yuhang Chen, Jiaxin Jiang, Shixuan Sun, Bingsheng He, and Min Chen. 2024. Rush: Real-time burst subgraph detection in dynamic graphs. *Proceedings of the VLDB Endowment* 17, 11 (2024), 3657–3665.

[28] Young-Rae Cho and Aidong Zhang. 2009. Predicting protein function by frequent functional association pattern mining in protein interaction networks. *Transactions on information technology in biomedicine* 14, 1 (2009), 30–36.

[29] Sutanay Choudhury, Lawrence Holder, George Chin, Khushbu Agarwal, and John Feo. 2015. A selectivity based approach to continuous pattern detection in streaming graphs. *arXiv preprint arXiv:1503.00849* (2015).

[30] Aaron Clauset, Cosma Rohilla Shalizi, and Mark EJ Newman. 2009. Power-law distributions in empirical data. *SIAM review* 51, 4 (2009), 661–703.

[31] Mukund Deshpande, Michihiro Kuramochi, Nikil Wale, and George Karypis. 2005. Frequent substructure-based approaches for classifying chemical compounds. *Transactions on Knowledge and Data Engineering* 17, 8 (2005), 1036–1050.

[32] Vinicius Dias, Carlos HC Teixeira, Dorgival Guedes, Wagner Meira, and Srinivasan Parthasarathy. 2019. Fractal: A general-purpose graph pattern mining system. In *SIGMOD*. 1357–1374.

[33] Reinhard Diestel. 2024. *Graph theory*. Springer (print edition); Reinhard Diestel (eBooks).

[34] David Easley, Jon Kleinberg, et al. 2010. *Networks, crowds, and markets: Reasoning about a highly connected world*. Vol. 1. Cambridge university press Cambridge.

[35] Mark S Granovetter. 1973. The strength of weak ties. *American journal of sociology* 78, 6 (1973), 1360–1380.

[36] Wentian Guo, Yuchen Li, Mo Sha, Bingsheng He, Xiaokui Xiao, and Kian-Lee Tan. 2020. Gpu-accelerated subgraph enumeration on partitioned graphs. In *SIGMOD*.

[37] Guyue Han and Harish Sethu. 2016. Waddling random walk: Fast and accurate mining of motif statistics in large graphs. In *ICDM*.

[38] Tomaž Hočevar and Janez Demšar. 2014. A combinatorial approach to graphlet counting. *Bioinformatics* 30, 4 (2014), 559–565.

[39] Paul W Holland and Samuel Leinhardt. 1976. Local structure in social networks. *Sociological methodology* 7 (1976), 1–45.

[40] Yang Hu, Hang Liu, and H Howie Huang. 2018. Tricore: Parallel triangle counting on gpus. In *SC*.

[41] Fanchen Bu Jihoon Ko Kijung Shin Hyunju Kim, Heechan Moon. 2025. Estimating simplet counts via sampling: Estimating simplet counts... *The VLDB Journal* 34, 2 (2025). https://doi.org/10.1007/s00778-024-00890-9

[42] Fanchen Bu Kijung Shin Hyunju Kim, Jihoon Ko. 2023. Characterization of Simplicial Complexes by Counting Simplets Beyond Four Nodes. In *Proceedings of the ACM Web Conference 2023*. 317–327. https://doi.org/10.1145/3543507.3583332

[43] Anand Padmanabha Iyer, Zaoxing Liu, Xin Jin, Shivaram Venkataraman, Vladimir Braverman, and Ion Stoica. 2018. ASAP: Fast, approximate graph pattern mining at scale. In *OSDI 18*.

[44] Kasra Jamshidi, Rakesh Mahadasa, and Keval Vora. 2020. Peregrine: a pattern-aware graph mining system. In *EuroSys*.

[45] Madhav Jha, C Seshadhri, and Ali Pinar. 2015. Path sampling: A fast and provable method for estimating 4-vertex subgraph counts. In *WWW*.

[46] John Kallaugher, Michael Kapralov, and Eric Price. 2018. The Sketching Complexity of Graph and Hypergraph Counting. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*. 556–567. https://doi.org/10.1109/FOCS.2018.00059

[47] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. 2010. What is Twitter, a social network or a news media?. In *WWW*.

[48] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford large network dataset collection.

[49] Dean Lusher, Johan Koskinen, and Garry Robins. 2013. *Exponential random graph models for social networks: Theory, methods, and applications*. Cambridge University Press.

[50] Daniel Mawhirter, Sam Reinehr, Connor Holmes, Tongping Liu, and Bo Wu. 2021. Graphzero: A high-performance subgraph matching system. *ACM SIGOPS Operating Systems Review* 55, 1 (2021), 21–37.

[51] Daniel Mawhirter and Bo Wu. 2019. Automine: harmonizing high-level abstraction and high performance for graph mining. In *SOSP*.

[52] Tijana Milenković, Weng Leong Ng, Wayne Hayes, and Nataša Pržulj. 2010. Optimal network alignment with graphlet degree vectors. *Cancer informatics* 9 (2010), CIN–S4744.

[53] Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. 2002. Network motifs: simple building blocks of complex networks. *Science* 298, 5594 (2002), 824–827.

[54] Richard Montgomery and Matías Pavez-Signé. 2024. Counting spanning subgraphs in dense hypergraphs. *Combinatorics, Probability and Computing* 33, 6 (2024), 729–741. https://doi.org/10.1017/S0963548324000178

[55] Ryan A. Rossi Nick G. Duffield Theodore L. Willke Nesreen K. Ahmed, Jennifer Neville. 2017. Graphlet Decomposition: Framework, Algorithms, and Applications. *Knowledge and Information Systems* 50, 3 (2017), 689–722. https://doi.org/10.1007/s10115-016-0965-5

[56] Mark EJ Newman. 2001. Clustering and preferential attachment in growing networks. *Physical review E* 64, 2 (2001), 025102.

[57] Mark EJ Newman. 2003. The structure and function of complex networks. *SIAM review* 45, 2 (2003), 167–256.

[58] Rasmus Pagh and Charalampos E Tsourakakis. 2012. Colorful triangle counting and a mapreduce implementation. *Inform. Process. Lett.* 112, 7 (2012), 277–281.

[59] Aduri Pavan, Kanat Tangwongsan, Srikanta Tirthapura, and Kun-Lung Wu. 2013. Counting and sampling triangles from a graph stream. *Proceedings of the VLDB Endowment* 6, 14 (2013), 1870–1881.

[60] Ali Pinar, Comandur Seshadhri, and Vaidyanathan Vishal. 2017. Escape: Efficiently counting all 5-vertex subgraphs. In *Proceedings of the 26th international conference on world wide web*. 1431–1440.

[61] Ryan A. Rossi, Nesreen K. Ahmed, Aldo Carranza, David Arbour, Anup Rao, Sungchul Kim, and Eunyee Koh. 2019. Heterogeneous Network Motifs. arXiv:1901.10026 [cs.SI] https://arxiv.org/abs/1901.10026

[62] Marc Roth and Johannes Schmitt. 2020. Counting Induced Subgraphs: A Topological Approach to W[1]-hardness. *Algorithmica* 82, 8 (2020), 2267–2291. https://doi.org/10.1007/s00453-020-00676-9

[63] Aldo Carranza David Arbour Anup Rao Sungchul Kim Eunyee Koh Ryan A. Rossi, Nesreen K. Ahmed. 2020. Heterogeneous Graphlets. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 15, 1, Article 9 (2020), 43 pages. https://doi.org/10.1145/3418773

[64] Tung Mai Nesreen K. Ahmed Ryan A. Rossi, Anup Rao. 2020. Fast and Accurate Estimation of Typed Graphlets. In *Companion Proceedings of the Web Conference 2020*. 32–34. https://doi.org/10.1145/3366424.338268

[65] Siddhartha Sahu, Amine Mhedhbi, Semih Salihoglu, Jimmy Lin, and M Tamer Özsu. 2017. The ubiquity of large graphs and surprising challenges of graph processing. *Proceedings of the VLDB Endowment* 11, 4 (2017), 420–431.

[66] Gennady Samorodnitsky and Takashi Owada. 2023. Large deviations for subcomplex counts and Betti numbers in multiparameter simplicial complexes. *Random Structures Algorithms* 63, 2 (2023), 533–556. https://doi.org/10.1002/rsa.21146

[67] Seyed-Vahid Sanei-Mehri, Ahmet Erdem Sariyuce, and Srikanta Tirthapura. 2018. Butterfly counting in bipartite networks. In *KDD*. 2150–2159.

[68] Fazle E. Faisal Shawn Gu, John Johnson and Tijana Milenković. 2018. From homogeneous to heterogeneous network alignment via colored graphlets. *Scientific Reports* 8, 12524 (2018).

[69] Jessica Shi, Laxman Dhulipala, and Julian Shun. 2021. Parallel clique counting and peeling algorithms. In *ACDA*.

[70] Jessica Shi, Louisa Ruixue Huang, and Julian Shun. 2022. Parallel Five-cycle Counting Algorithms. *Journal of Experimental Algorithmics* 27 (2022), 1–23.

[71] Tianhui Shi, Mingshu Zhai, Yi Xu, and Jidong Zhai. 2020. Graphpi: High performance graph pattern matching through effective redundancy elimination. In *SC*.

[72] Wonseok Shin, Siwoo Song, Kunsoo Park, and Wook-Shin Han. 2024. Cardinality Estimation of Subgraph Matching: A Filtering-Sampling Approach. *Proceedings of the VLDB Endowment* 17, 7 (2024), 1697–1709. https://arxiv.org/abs/2309.15433

[73] Julian Shun and Kanat Tangwongsan. 2015. Multicore triangle computations without tuning. In *ICDE*.

[74] Honglong Chen Ivan Lee Lianhua Chi Hanghang Tong Shuo Yu, Feng Xia. 2024. Heterogeneous network motif coding, counting, and profiling. *ACM Transactions on Knowledge Discovery from Data* 18, 9 (2024), Article 231. https://doi.org/10.1145/3687465

[75] George M Slota and Kamesh Madduri. 2013. Fast approximate subgraph counting and enumeration. In *ICPP*.

[76] Yuhang Su, Yu Gu, Zhigang Wang, Ying Zhang, Jianbin Qin, and Ge Yu. 2023. Efficient Subhypergraph Matching Based on Hyperedge Features. *IEEE Transactions on Knowledge and Data Engineering* 35, 6 (2023), 5808–5822. https://doi.org/10.1109/TKDE.2022.3160393

[77] Carlos HC Teixeira, Alexandre J Fonseca, Marco Serafini, Georgos Siganos, Mohammed J Zaki, and Ashraf Aboulnaga. 2015. Arabesque: a system for distributed graph mining. In *SOSP*.

[78] Charalampos E Tsourakakis, U Kang, Gary L Miller, and Christos Faloutsos. 2009. Doulion: counting triangles in massive graphs with a coin. In *KDD*.

[79] Chad Voegele, Yi-Shan Lu, Sreepathi Pai, and Keshav Pingali. 2017. Parallel triangle counting and k-truss identification using graph-centric methods. In *HPEC*.

[80] Jozef Skokan Vojtěch Rödl. 2005. Counting subgraphs in quasi-random 4-uniform hypergraphs. *Random Structures Algorithms* 26, 1-2 (2005), 160–203. https://doi.org/10.1017/S0963548324000178

[81] Kaixin Wang, Cheng Long, Da Yan, Jie Zhang, and HV Jagadish. 2023. Reinforcement learning enhanced weighted sampling for accurate subgraph counting on fully dynamic graph streams. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 1084–1097.

[82] Kai Wang, Zhiqiang Zuo, John Thorpe, Tien Quang Nguyen, and Guoqing Harry Xu. 2018. RStream: Marrying relational algebra with streaming for efficient graph mining on a single machine. In *OSDI*.

[83] Pinghui Wang, John CS Lui, Bruno Ribeiro, Don Towsley, Junzhou Zhao, and Xiaohong Guan. 2014. Efficiently estimating motif statistics of large networks. *Transactions on Knowledge Discovery from Data* 9, 2 (2014), 1–27.

[84] Duncan J Watts. 1999. Small worlds: the dynamics of networks between order and randomness.

[85] Duncan J Watts and Steven H Strogatz. 1998. Collective dynamics of 'small-world'networks. *nature* 393, 6684 (1998), 440–442.

[86] Xifeng Yan and Jiawei Han. 2002. gspan: Graph-based substructure pattern mining. In *ICDM*.

[87] Carl Yang, Mengxiong Liu, Vincent W Zheng, and Jiawei Han. 2018. Node, motif and subgraph: Leveraging network functional blocks through structural convolution. In *ASONAM*.

[88] Jaewon Yang and Jure Leskovec. 2012. Defining and evaluating network communities based on ground-truth. In *SIGKDD workshop on mining data semantics*.

[89] Zhengyi Yang, Wenjie Zhang, Xuemin Lin, Ying Zhang, and Shunyang Li. 2023. HGMatch: A Match-by-Hyperedge Approach for Subgraph Matching on Hypergraphs. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. 2063–2076. https://doi.org/10.1109/ICDE55515.2023.00160

[90] Zhao Zhao, Maleq Khan, VS Anil Kumar, and Madhav V Marathe. 2010. Subgraph enumeration in large social contact networks using parallel color coding and streaming. In *ICPP*.

[91] Zeying Zhu, Kan Wu, and Zaoxing Liu. 2023. Arya: Arbitrary Graph Pattern Mining with Decomposition-based Sampling. In *NSDI*.