AGS-GNN: Attribute-guided Sampling for Graph Neural Networks

Siddhartha Shankar Das
das90@purdue.edu
Purdue University
West Lafayette, IN, USA

S M Ferdous
sm.ferdous@pnnl.gov
Pacific Northwest National Lab.
Richland, WA, USA

Mahantesh M Halappanavar
hala@pnnl.gov
Pacific Northwest National Lab.
Richland, WA, USA

Edoardo Serra
edoardoserra@boisestate.edu
Boise State University
Boise, ID, USA

Alex Pothen
apothen@purdue.edu
Purdue University
West Lafayette, IN, USA

Abstract

We propose AGS-GNN, a novel attribute-guided sampling algorithm for Graph Neural Networks (GNNs). AGS-GNN exploits the node features and the connectivity structure of a graph while simultaneously adapting for both homophily and heterophily in graphs. In homophilic graphs, vertices of the same class are more likely to be adjacent, but vertices of different classes tend to be adjacent in heterophilic graphs. GNNs have been successfully applied to homophilic graphs, but their utility to heterophilic graphs remains challenging. The state-of-the-art GNNs for heterophilic graphs use the full neighborhood of a node instead of sampling it, and hence do not scale to large graphs and are not inductive. We develop dual-channel sampling techniques based on feature-similarity and feature-diversity to select subsets of neighbors for a node that capture adaptive information from homophilic and heterophilic neighborhoods. Currently, AGS-GNN is the only algorithm that explicitly controls homophily in the sampled subgraph through similar and diverse neighborhood samples. For diverse neighborhood sampling, we employ submodularity, a novel contribution in this context. We pre-compute the sampling distribution in parallel, achieving the desired scalability. Using an extensive dataset consisting of 35 small ($< 100K$ nodes) and large ($\geq 100K$ nodes) homophilic and heterophilic graphs, we demonstrate the superiority of AGS-GNN compared to the state-of-the-art approaches. AGS-GNN achieves test accuracy comparable to the best-performing heterophilic GNNs, even outperforming methods that use the entire graph for node classification. AGS-GNN converges faster than methods that sample neighborhoods randomly, and can be incorporated into existing GNN models that employ node or graph sampling.

CCS Concepts

• **Computing Methodologies** → **Machine Learning**.

Keywords

Graph Neural Networks, Heterophily, Submodular Functions

KDD '24, August 25–29, 2024, Barcelona, Spain

ACM ISBN 979-8-4007-0490-1/24/08
https://doi.org/10.1145/3637528.3671940

ACM Reference Format:
Siddhartha Shankar Das, S M Ferdous, Mahantesh M Halappanavar, Edoardo Serra, and Alex Pothen. 2024. AGS-GNN: Attribute-guided Sampling for Graph Neural Networks. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '24), August 25–29, 2024, Barcelona, Spain.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3637528.3671940

1 Introduction

Traditional Graph Neural Networks (GNNs) rely on the *homophilic* property of the learning problem, which assumes that a significant number of a node's neighbors share the class label of the node. However, this assumption has been challenged in recent years since graphs in several practical applications do not satisfy homophily [31, 49]. Consequently, GNNs designed with the assumption of homophily fail to classify heterophilic graphs accurately due to noisy or improper neighborhood aggregation [25]. A simple Multi-layer Perceptron (MLP) based model that ignores the graph structure can outperform existing homophilic GNNs on heterophilic graphs [8, 23, 24, 26, 45, 51]. As a result, a number of special-purpose GNNs have been developed to classify heterophilic graphs [8, 21, 24–26, 42, 43, 49–51].

Although GNNs have been adapted for heterophilic graphs in earlier work, their applicability is limited since they do not scale to large graphs and are transductive. Unlike homophilic GNNs [3, 4, 6, 12, 15, 46], where subgraph sampling strategies have been developed for scaling, currently there are no effective sampling approaches for heterophilic graphs [23]. Recent authors have enabled scaling by first transforming node features and adjacency matrix into lower dimensional representations, and then applying mini-batching on the combined representations [22, 23]. Thus inference on heterophilic graphs via graph sampling remains challenging.

Classifying graphs into either heterophilic or homophilic graphs, as done in the current literature, is problematic due to the presence of locally homophilic and locally heterophilic nodes. As shown in Fig. 1 (§2), both homophilic and heterophilic graphs could have nodes with high local homophily or heterophily. However, there is no systematic and scalable approach for neighborhood sampling that distinguishes each node w.r.t. its local homophily property. We propose a new sampling strategy that incorporates both the adjacency structure of the graph as well as the feature information of the nodes that is capable of making this distinction. For a homophilic node, we build our sampling strategy based on a widely

used *smoothing* assumption [36]: labels and features generally correlate positively. Heterophilic nodes of the same labels, however, are expected to have the same dissimilar neighborhood [25], and our sampling strategy exploits this. Thus, we generate two sets of local neighborhood samples for each node: one based on feature similarity that potentially improves local homophily, while the other encourages diversity and increases heterophily. These two samples are adaptively learned using an MLP to select the appropriate one based on the downstream task. Our attribute-based sampling strategy can be seamlessly plugged into GNNs designed for classifying heterophilic graphs and performs well in practice. The strength of our approach, however, is that even when paired with GNNs designed for homophilic graphs, we obtain better accuracies for heterophilic graphs, thus rendering an overall scalable approach for the latter graphs. The key contributions and findings of this work are summarized as follows:

(1) We propose a novel scalable and inductive unsupervised and supervised feature-guided sampling framework, AGS-GNN, to learn node representations for both homophilic and heterophilic graphs.

(2) AGS-GNN incorporates sampling based on similarity and diversity (modeled by a submodular function). We are not aware of earlier work that uses submodularity in this context. AGS-GNN employs dual channels with MLPs to learn from both similar and diverse neighbors of a node.

(3) We experimented with 35 benchmark datasets for node classification and compared them against GNNs designed for heterophilic and homophilic graphs. For both types of graphs, AGS-GNN achieved improved accuracies relative to earlier methods (Fig. 6) (§5.3). Further, AGS-GNN also requires fewer iterations (up to 50% less) (§5.4) to converge relative to random sampling.

2 Preliminaries

Consider a weighted graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ with set of vertices $\mathcal{V}$ and set of edges $\mathcal{E}$. We denote the number of vertices and edges by $|\mathcal{V}|$ and $|\mathcal{E}|$. The adjacency matrix of the graph will be denoted by $\boldsymbol{A} \in \mathbb{R}^{|\mathcal{V}|\times|\mathcal{V}|}$, and the matrix of f-dimensional feature vectors associated with nodes by $\boldsymbol{X} \in \mathbb{R}^{|\mathcal{V}|\times f}$. We denote $y_u \in \mathbb{Y} = \{1, 2, \cdots, c\}$ to be the label of a node u, which belongs to one of the c classes, and the vector $\boldsymbol{y} \in \mathbb{Y}^{|\mathcal{V}|}$ to denote the labels of all nodes. Additionally, the graph may have edge features associated with dimension f_e. The degree of node u is denoted by d_u, the average degree by d, and the set $\mathcal{N}(u)$ denotes the set of neighboring vertices of a node u. For a GNN, ℓ denotes the number of layers, H the number of neurons in the hidden layer, and $\boldsymbol{W}^i$ the learnable weight of the i-th layer.

2.1 Homophily Measures

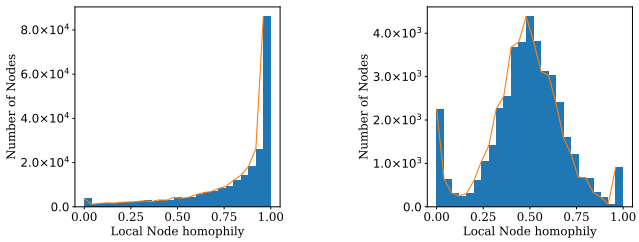
The *homophily* of a graph characterizes how likely it is for vertices with the same labels to be neighbors of each other. Many measures of homophily fit this definition. But for conciseness, we will focus here on *node homophily* ($\mathcal{H}_n$) (intuitive), and *adjusted homophily* ($\mathcal{H}_a$) (handles class imbalance). The *local node homophily* of node u is $\mathcal{H}_n(u) = \frac{|\{v \in \mathcal{N}(u) : y_v = y_u\}|}{|\mathcal{N}(u)|}$, and its mean value *node homophily* [30] is defined as follows:

$$\mathcal{H}_n = \frac{1}{|\mathcal{V}|} \sum_{u \in \mathcal{V}} \mathcal{H}_n(u). \tag{1}$$

The *edge homophily* [51] of a graph is $\mathcal{H}_e = \frac{|\{(u,v) \in \mathcal{E} : y_u = y_v\}|}{|\mathcal{E}|}$. Let $D_k = \sum_{v : y_v = k} d_v$ denote the sum of degrees of the nodes belonging to class k. Then the *adjusted homophily* [31] is defined as

$$\mathcal{H}_a = \frac{\mathcal{H}_e - \sum_{k=1}^{c} D_k^2/(2|\mathcal{E}|^2)}{1 - \sum_{k=1}^{c} D_k^2/2|\mathcal{E}|^2}. \tag{2}$$

The values of the node homophily and edge homophily range from 0 to 1, and the adjusted homophily ranges from $-\frac{1}{3}$ to +1 (Proposition 1 in [31]). In this paper, we will classify graphs with adjusted homophily less than 0.50 as heterophilic. Fig. 1 shows the distribution of the *local node homophily* of a homophilic and a heterophilic graph. We see that both the graphs have a mix of locally heterophilic and locally homophilic nodes.



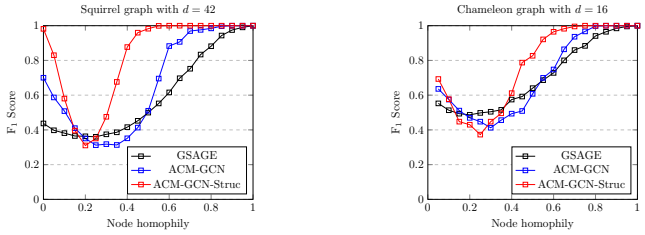
(a) Reddit (homophilic) (b) Penn94 (heterophilic)

Figure 1: The distribution of local node homophily in a homophilic and a heterophilic graph.

2.2 Effect of Homophily on Classification

To highlight the effect of homophily on node classification, we conduct an experiment using synthetic graphs with different levels of node homophily. These synthetic graphs are generated from existing graphs by ignoring the structure information of the graph but retaining the node features and class labels. Following [25], to generate an undirected graph with an average degree of d and node homophily $\mathcal{H}_n$, for each node u we randomly assign $\mathcal{H}_n \cdot d/2$ edges from the same class as u and $(1 - \mathcal{H}_n) \cdot d/2$ edges from other classes. We left the class distribution unbalanced, as it is in the original graph, making it more challenging for GNNs since the neighborhood of a heterophilic node could potentially have more nodes from the majority class.

Fig. 2 shows the performance of GSAGE [12] (a homophilic GNN) and ACM-GCN [25] (a heterophilic GNN) on the synthetic graphs generated from Squirrel and Chameleon [33] datasets. We use two versions of ACM-GCN, one with three channels (low-pass, high-pass, and identity) and the other with an additional channel with graph structure information (ACM-GCN-struc). The original Squirrel and Chameleon datasets are heterophilic, and the ACM-GCN-struc is the best-performing. For synthetic graphs, on both Squirrel and Chameleon, (Fig. 2), we see that surprisingly the worst F_1 score is not achieved on the graphs whose homophily value is zero but for values near 0.25. When the homophily score is high, GNNs perform well since they aggregate relevant information, but as we observe ACM-GCN also does well at a very low homophily. This is because some locally heterophilic nodes become easier to classify after neighborhood aggregation on features. (note that



(a) Squirrel Synthetic **(b) Chameleon Synthetic**

Figure 2: F_1 Score comparison of GSAGE and ACM-GCN on synthetic graphs generated from Squirrel (a) and Chameleon (b) datasets with varying node homophily.

features are not considered in the definition of homophily based solely on labels). Another intuitive reason is that when two nodes are adjacent to the same dissimilar (wrt class labels) neighbors, the high pass filters (e.g., graph Laplacian) used in ACM-GCN treat these nodes as similar and can classify them correctly.

2.3 Similarity, Diversity, and Homophily

Nodes with similar features tend to have similar labels [36]. Therefore, if instead of sampling the neighbors of a node u uniformly at random, we sample neighbors that are similar to u in feature space, we are likely to increase the homophily of the sampled subgraph.

However, this strategy alone is not enough for heterophilic graphs as they include both locally homophilic and heterophilic nodes (Fig. 1). Two heterophilic nodes with the same label are expected to have similar class label distributions in their neighbors. In other words, the diversity in their class labels makes them similar. It has been shown [25, 42, 49] that high-pass filters [9] (e.g., variants of graph Laplacians: $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{A}$, $\boldsymbol{L}_{sym} = \boldsymbol{D}^{-1/2}\boldsymbol{L}\boldsymbol{D}^{-1/2}$, or $\boldsymbol{L}_{rw} = \boldsymbol{D}^{-1}\boldsymbol{L}$) capture the difference between nodes, and low-pass filters (e.g., scaled adjacency matrices, $\boldsymbol{A}_{sym} = \boldsymbol{D}^{-1/2}\boldsymbol{A}\boldsymbol{D}^{-1/2}$, or $\boldsymbol{A}_{rw} = \boldsymbol{D}^{-1}\boldsymbol{A}$) retain the commonality of node features. Here $\boldsymbol{D}$ is the diagonal degree matrix of the graph. Therefore, if we sample diverse neighbors (in feature space) and use a GNN with a high-pass filter, we expect a higher chance of mapping two heterophilic nodes of same class to the same space (after feature transformation) since they will have the same dissimilar neighborhood.

A mathematical approach to ensure diversity is through *submodular function* maximization [20, 34]. A submodular function is a set function that favors a new node that is most distant in feature space to add to a partially sampled neighborhood. It accomplishes this by defining a suitable marginal gain of the new element with respect to the current neighborhood, and maximizing this value. However, employing only diverse neighborhoods can also cause issues since two nodes with different labels may have similar neighborhoods after sampling. In this scenario, sampling based on similarity is more appropriate. For the *spectral* domain, AGS-GNN considers two channels: one with a sampled subgraph ensuring diversity (used with a high-pass filter) and the other with a subgraph sampled based on similarity (used with a low-pass filter). Similar to ACM-GCN, we can also use an identity channel. However, *spectral* GNNs are difficult to scale as they often do not support mini-batching, and are transductive. Hence, we consider *spatial* GNNs for heterophilic graphs, which employ graph topology to develop aggregation strategies. For heterophilic graphs, both similar and dissimilar neighbors need to be considered, and in AGS-GNN, we achieve this through attribute-guided biased sampling of similar and diverse samples.

2.3.1 Node Homophily with Similar and Diverse neighborhood: Consider an ego node $t = \{\boldsymbol{x}_t, y_t\}$ with feature $\boldsymbol{x}_t$, label y_t, and local node homophily $\mathcal{H}_n(t)$. Let the feature and label tuples of the neighbors of t be $\mathcal{N}(t) = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_{d_t}, y_{d_t})\}$. From the definition of homophily, the probability of randomly selecting a neighbor $i \in \mathcal{N}(t)$ with the same label as the ego node is $P_{\mathcal{U}}(y_i = y_t) = \mathcal{H}_n(t)$. Here $\mathcal{U}$ refers to the distribution of selecting a neighbor uniformly at random. Let $s(\boldsymbol{x}_i, \boldsymbol{x}_t)$ be a positive similarity score in features between a neighbor, i, and the ego node, t.

Assumption 2.1. *For a node t, the average similarity of neighbors with the same label as t is greater than or equal to the average similarity of all neighbors.*

Lemma 2.1. *With Assumption 2.1, if the probability of selecting a neighboring node is proportional to its similarity to the ego node t, the local node homophily of sampled neighborhood $\mathcal{H}'_n(t) \geq \mathcal{H}_n(t)$. If the sampling probability distribution is $\mathcal{S}$, then $P_{\mathcal{S}}(y_i = y_t) \geq P_{\mathcal{U}}(y_i = y_t)$.*

To retrieve a diverse set of neighbors we can employ a submodular function. An example could be the *facility location* function based on maximizing pairwise similarities between the points in the data set and their nearest chosen point, $f(S, A) = \sum_{y \in A} \max_{x \in S} \phi(x, y)$, where A is the ground set, $S \subseteq A$ is a subset, and ϕ is the similarity measure. In our context, S is the current set of selected nodes initialized with ego node t, and ground set $A = \mathcal{N}(t) \cup \{t\}$. The *marginal gain* is $f_i = f(S \cup \{i\}, A) - f(S, A)$ for each neighbor $i \in \mathcal{N}(t) \setminus S$. Successively, neighbors are added to the sampled neighborhood by choosing them to have maximum marginal gain with respect to the current sample of the neighborhood.

Assumption 2.2. *The average marginal gain of the neighbors of a node t with the same label as t is less than or equal to the average marginal gain of all neighbors.*

Lemma 2.2. *With Assumption 2.2, if the probability of selecting a neighboring node is proportional to its marginal gain wrt the ego node t, then the local node homophily of sampled neighborhood $\mathcal{H}'_n(t) \leq \mathcal{H}_n(t)$. If the sampling probability distribution is $\mathcal{D}$, then $P_{\mathcal{D}}(y_i = y_t) \leq P_{\mathcal{U}}(y_i = y_t)$.*

Proofs of Lemmas 2.1 and 2.2 are included in Appendix 7.1 and Appendix 7.2, respectively. From Lemma 2.1, sampling neighbors based on feature-similarity improves homophily, which potentially helps to map homophilic nodes of the same labels into the same space. We assume features and labels to be positively correlated; thus, if we ensure feature diversity among neighbors, we can also expect label diversity in the samples, reducing local homophily (as shown in Lemma 2.2) and increasing the chances of mapping two heterophilic nodes into the same space. We devise feature-similarity and feature-diversity-based sampling based on these results in the next Section.

3 Proposed Method: AGS-GNN

Fig. 3 shows an overview of the AGS-GNN framework. AGS-GNN has a *pre-computation* step that ranks the neighbors of nodes and computes edge weights to form a distribution to sample from in the *sampling* phases during training.

3.1 Pre-computing Probability Distribution

An ideal graph sampling process has the following requirements: (i) Nodes with strong mutual influences (in terms of structure and attributes) should be sampled together in a subgraph [46]. (ii) It should be able to distinguish between similar and dissimilar neighbors (especially for heterophilic graphs) [49]. (iii) Every edge should have a non-zero probability of being sampled in order to generalize and explore the full feature and label space.

In this Section, we will devise sampling strategies satisfying these requirements. We assume that we have access to a similarity measure between any two nodes in the graph. This similarity function typically depends on the problem and the dataset. For an example, in text based datasets, we may use cosine similarity of the feature vectors (e.g., TF-IDF) generated from the texts of the corresponding items. We may also learn the similarities when an appropriate similarity measure is not apparent. Once we have the similarity function, for each vertex u, we construct a probability distribution over its neighbors as follows: (i) *rank* the neighboring nodes ($\mathcal{N}(u)$) using their similarity scores to u. (ii) assign weights to the adjacent edges of u based on this ranking. (iii) normalize the weights to construct the probability mass function (PMF) $\mathcal{P}(u)$ of the distribution over $\mathcal{N}(u)$.

We construct the probability distribution using the rank rather than the actual similarity values, since the distribution using the similarity values could be skewed and extremely biased towards a few top items. Here we consider two choices of rankings of neighborhoods that are suited for homophilic and heterophilic graphs.

3.1.1 Ranking based on similarity: For this case, our goal is to sample subgraphs favoring similar edges to be present more frequently in the subgraph. To achieve that we propose to construct a probability distribution over the edges based on the similarity scores. We sort the similarity values of all the neighboring vertices of a vertex from high to low and rank adjacent edges based on the ranks. Thus, although the similarity function is symmetric, we may get two different ranks for each edge. Note that the computation that generates the ranks is local to each vertex and thus highly parallel. For cosine similarity, the time complexity to rank the adjacent edges of a vertex u is $O(fd_u \log d_u)$.

Once we have the rankings, we can use these to construct different probability distributions. Some choices (Fig. 4) of Probability Mass Functions (PMF) can be *linear* or *exponential* decay with non-zero selection probability to the later elements in the ranking order. Another options is the *step* function where the top k_1% neighbors of a vertex are given a uniform weight (λ_1), the next k_2% are given λ_2, and the rest are given λ_3, where $\lambda_1 > \lambda_2 > \lambda_3$. The benefit of using such function is that we can partially sort the top $(k_1 + k_2)$% of neighbors avoiding the full ordering. Algorithm 1 shows how to get sampling probabilities using rank by similarity.

3.1.2 Ranking based on diversity: As discussed in Section 2.3, for a heterophilic graph, in general it is desirable to construct subgraphs based on diversity in the class labels. To accomplish that, we propose a sampling strategy based on optimizing (a nonlinear) submodular function. A submodular function is a set function that has the diminishing returns property. Formally, given a ground set V and two subsets A and B with $A \subseteq B \subseteq V$, a function f is submodular if and only if, for every $e \in V \backslash B$, $f(A \cup e) - f(A) \geq f(B \cup e) - f(B)$. The quantity on either side of the inequality is called the *marginal gain* of an element e with respect to the two sets A or B. For maximizing cardinality constrained submodular functions, where the solution set (S) is required to be at most k in size, a natural *Greedy* algorithm that starts with empty solution and augments it with the element with highest marginal gain is $(1 - 1/e)$-approximate [28].

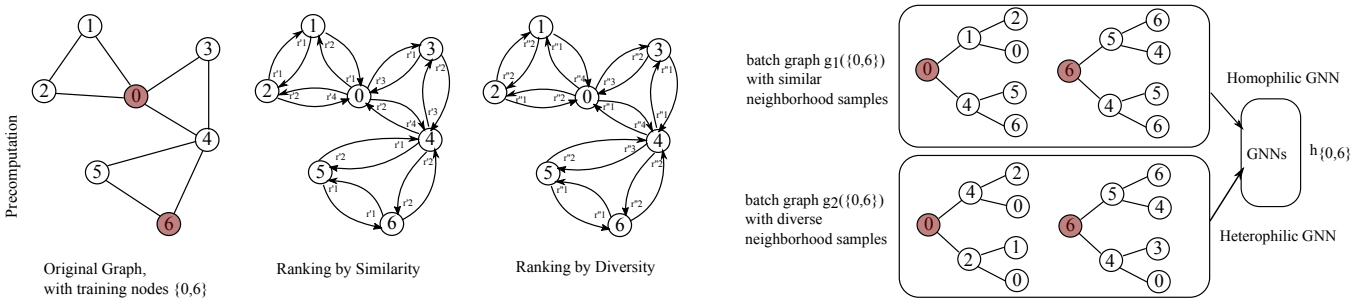
To see how a submodular function may behave differently than the (linear) similarity based function (Section 3.1.1), consider a paper citation graph [35] where the nodes are the scientific documents and the feature on each node is the binary count vector of the associate text. For a vertex u, our goal is to find a set of k neighboring vertices of u, ($S \subseteq \mathcal{N}(u)$ where $|S| = k$), such that given the initial set $\{u\}$, we maximize the number of *unique* word counts. This objective can be modeled as a submodular function known as maximum k-coverage, and the Greedy approach would select successive nodes with maximum marginal gains wrt to S. Intuitively, the Greedy algorithm prioritizes neighbors that have more distinct words than the ones covered through the selected nodes. Therefore, if different word sets correspond to different class labels, the final set S will likely represent a diverse set. This contrasts sharply with the ranking based on similarity, where we would encourage neighbors with similar words. *Facility Location, Feature-based functions*, and *Maximum Coverage* are some submodular functions applicable in the graph context.

Given a submodular function, for a vertex u, we execute the Greedy algorithm on the neighbors of u to compute their marginal gains, assuming u is in the initial solution. We use these marginal gains to rank neighbors of u and then use the ranks to construct a probability distribution as described in Section 3.1.1. To compute the solution faster, we employed a variant of the Greedy algorithm which is called Lazy Greedy [27] that can reduce the number of marginal gain computations. Algorithms 2 and 3 show the pseudocode of the Lazy Greedy version of the ranking procedure using the *facility location* function. Since we need to compute pairwise similarity for this function, the complexity of computing the ranks of the neighbors of a vertex u is $O(fd_u^2)$.

Algorithm 1 RankBySimilarity($\mathcal{G}, \boldsymbol{X}, \mathcal{S}, \mathcal{P}$, *params)

Input: Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, Feature matrix $\boldsymbol{X} \in \mathbb{R}^{|\mathcal{V}| \times f}$, Similarity function $\mathcal{S}$, Probability Mass Function, $\mathcal{P}$

1: **for** $u \in \mathcal{V}$ **do**
2: $\mathcal{N}(u)$ = GetNeighbors($\mathcal{G}, u$) /* Get neighbors of vertex u */
3: $S_u = \mathcal{S}(\boldsymbol{X}[u], \boldsymbol{X}[\mathcal{N}(u)])$ /* Similarities from u to $\mathcal{N}(u)$ */
4: R_u = Rank(N_u, S_u) /* Rank $\mathcal{N}(u)$ in desc. order of S_u */
5: W_u = GetWeights($R_u, \mathcal{P}$) /* Assign weights $W_{u,v} : v \in \mathcal{N}(u)$ from the ranks R_u using $\mathcal{P}$ */
6: $P_{u,v} = W_{u,v} / \sum_{v \in \mathcal{N}(u)} W_{u,v}$ /* Edge sampling probability */
7: **end for**
8: **return** P



(a) Rank of neighbors for similar and diverse sampling. **(b) Node Sampling from training vertices with hop size 2**

Figure 3: AGS-GNN framework with Node Sampling. a) In the pre-computation step, ranks are computed based on feature similarity and diversity, and each edge has two ranks based on each vertex's perspective. b) The probability mass function is used to get sampling probabilities from ranks, and the figure demonstrates how two types of weighted neighborhood sampling are computed from the training nodes.

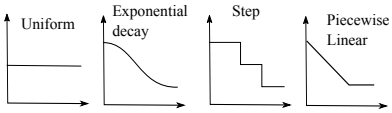


Figure 4: Probability Mass Functions (PMFs) for weights used later for selection probabilities.

Algorithm 2 RankByDiversity ($\mathcal{G}, \boldsymbol{X}, \mathcal{V}, \mathcal{S}, \mathcal{P}, *$params)

Input : Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, Feature matrix $\boldsymbol{X} \in \mathbb{R}^{|\mathcal{V}| \times f}$, Similarity function $\mathcal{S}$, Probability Mass Function, $\mathcal{P}$

for $u \in \mathcal{V}$ **do**
  $\mathcal{N}(u)$ = GetNeighbors($\mathcal{G}, u$) /* Get neighbors of vertex u */
  R_u = LazyGreedy($\mathcal{N}(u), u, \boldsymbol{X}_u | \boldsymbol{X}_{\mathcal{N}(u)}$) /* Rank $\mathcal{N}(u)$ */
  W_u = GetWeights($R_u, \mathcal{P}$) /* Assign weights $W_{u,v} : v \in \mathcal{N}(u)$ from the ranks R_u using $\mathcal{P}$ */
  $P_{u,v} = W_{u,v} / \sum_{v \in \mathcal{N}(u)} W_{u,v}$ /* Edge sampling probability */
end for
return P

3.1.3 Learnable Similarity Function: When we lack domain knowledge to compute an appropriate similarity metric, we can use the training subgraph to train a regression model to learn the edge weights of the graphs [8]. We form training batches with equal numbers of edges and non-edges randomly chosen from the training subgraph. We set the target label as 1 for nodes with the same labels and 0 otherwise, and then train a model using the batches to get an approximation of edge weights. We use these weights as similarity values to compute ranks based on similarity or diversity (as detailed in Section 3.1.1 and Section 3.1.2), and to compute a probability distribution over the directed edges. Given the number of training edges and an equal number of non-edges, we can expect the computation complexity to be $O(f|\mathcal{E}|)$. The neural network model is shallow, learnable parameters are relatively small, and computation is performed in batches, making it very fast. Algorithm 4 shows training procedure for a Siamese [7] edge-weight computation model SIM_W.

Algorithm 3 LazyGreedy ($\mathcal{N}(u), u, \boldsymbol{X}', \mathcal{S}$)

Input: $\mathcal{N}(u)$ is the neighbors of ego node u, $\boldsymbol{X}' \in \mathbb{R}^{|\mathcal{N}(u) \cup \{u\}|}$ is the feature of $\mathcal{N}(u) \cup \{u\}$, Similarity function $\mathcal{S}$

kernel = compute_pairwise_similarity($X', \mathcal{S}$)
$S = \{u\}$ /* Initialize set with u */
$Y = \{u\} \cup \mathcal{N}(u)$ /* Initialize Ground set */
S_{gain} = Gain(S, Y, kernel)
H = MaxHeap($S_{\text{gain}}, \mathcal{N}(u)$) /* Initialize max-heap H with key $\in S_{\text{gain}}$ and element $\in \mathcal{N}(u)$ */
while $H \neq \emptyset$ **do**
  (gain, v) = H.pop()
  gain_v = Gain($S \cup \{v\}, Y$, kernel) − S_{gain}
  ($\text{gain}_2, \cdot$) = H.top()
  if $\text{gain}_v \geq \text{gain}_2$ **then**
    $S = S \cup \{v\}$
    S_{gain} = Gain(S, Y, kernel)
  else
    H.push(gain_v, v)
  end if
end while
R_u = Rank(S) /* Rank the vertices by marginal gain */
return R_u

Algorithm 4 LearnSimilarity ($\mathcal{G}, \boldsymbol{X}, \boldsymbol{y}$)

Input: Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, Feature matrix $\boldsymbol{X} \in \mathbb{R}^{|\mathcal{V}| \times f}$, Label $\boldsymbol{y} \in \mathbb{Y}^{|\mathcal{V}|}$

$\mathcal{G}'$ = Subgraph($\mathcal{G}, \boldsymbol{y}$) /* Extract training subgraph */
$\text{SIM}_{\boldsymbol{W}}$ = Model(f, H, ℓ) /* Initialize a Siamese model */
for epoch **in** num_epochs **do**
  B = Batch($\mathcal{G}'$) /* Create batch with equal number edges and non-edges */
  $T = [(y_u = y_v) : (u, v) \in B]$ /* Create target label */
  $\text{SIM}_{\boldsymbol{W}}$ = Train(B, T) /* Train $\text{SIM}_{\boldsymbol{W}}$ using B, T */
end for
$\mathcal{E}' = \text{SIM}_{\boldsymbol{W}}(\mathcal{E})$ /* Predict edge weight */
return $\text{SIM}_{\boldsymbol{W}}, \mathcal{E}'$

3.2 Sampling

Algorithm 5 describes a node sampling process for training, AGS-NS, that is used in the AGS-GNN framework. The method *NodeSample*

is employed for sampling an ℓ-hop neighborhood based on the probability distribution derived from learned or computed weights.

Algorithm 5 AGS-NS $(\mathcal{G}, \boldsymbol{X}, \boldsymbol{y})$ [Node Sampling]

Input: Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, Feature matrix $\boldsymbol{X} \in \mathbb{R}^{|\mathcal{V}| \times f}$, Label, $\boldsymbol{y} \in \mathbb{Y}^{|\mathcal{E}|}$

$\mathcal{S}$ = LearnSimilarity$(\mathcal{G}, \boldsymbol{X}, \boldsymbol{y})$
$\mathcal{P}$ = 'step' /* probability mass function */
R_1 = RankBySimilarity$(\mathcal{G}, \boldsymbol{X}, \mathcal{S}, \mathcal{P})$/* Alg. 1 */
R_2 = RankByDiversity$(\mathcal{G}, \boldsymbol{X}, \mathcal{S}, \mathcal{P})$/* Alg. 2 */
for epoch **in** num_epochs **do**
  nodes, $\boldsymbol{y}_{\text{nodes}}$ = BatchOfTrainingNodes$(\mathcal{G}, \boldsymbol{y})$ /* random nodes from training vertices */
  g_1 = NodeSample(R_1, nodes) /* similar samples */
  g_2 = NodeSample(R_2, nodes) /* diverse samples */
  output = GNN$_{\boldsymbol{W}}(g_1, g_2)$
  l = loss(output, $\boldsymbol{y}_{\text{nodes}}$)
  Backward propagate from l to update $\boldsymbol{W}$
end for

AGS can be plugged in with other GNNs that use a sampling strategy. Here, we demonstrate this with GSAGE and call it AGS-GSAGE. The original formulation of GSAGE for i-th layer is,

$$\boldsymbol{h}_v^{(i)} = \sigma(\boldsymbol{W}^{(i)}, \boldsymbol{h}_v^{(i-1)} | \text{AGG}(\{\boldsymbol{h}_u^{(i-1)} : u \in \mathcal{N}_{rn}(v)\}). \quad (3)$$

Here $\boldsymbol{h}$ denotes feature vectors, $\mathcal{N}_{rn}(v)$ ($\subseteq \mathcal{N}(v)$) is the subset of neighbors of v (of size k) sampled uniformly at random. AGG is any permutation invariant aggregation function (e.g., `mean, sum, max, LSTM`, etc.).

In AGS-GSAGE, we use the probability distribution derived in Section 3.1 to sample with or without replacement from the neighborhood of a node. Depending on the nature of the graph, single or dual-channel GNNs might be necessary. Some homophilic graphs may have only a few locally heterophilic nodes, where samples from a distribution generated by similarity only may be sufficient. However, we expect the dual-channel AGS to perform better for heterophilic graphs since they typically have both locally homophilic and heterophilic nodes.

Fig. 5 shows some possible computation graphs. The dual-channel AGS mechanism can be incorporated easily into GSAGE. We generate two similar and diverse neighborhood samples for the target node, compute the transformed feature representations using both samples and use MLPs to combine these representations.

$$\boldsymbol{h}_{v_{sim}}^{(i)} = \sigma(\boldsymbol{W}_{sim}^{(i)}, \boldsymbol{h}_v^{(i-1)} | \text{AGG}(\{\boldsymbol{h}_u^{(i-1)} : u \in \mathcal{N}_{sim}(v)\}) \quad (4)$$

$$\boldsymbol{h}_{v_{div}}^{(i)} = \sigma(\boldsymbol{W}_{div}^{(i)}, \boldsymbol{h}_v^{(i-1)} | \text{AGG}(\{\boldsymbol{h}_u^{(i-1)} : u \in \mathcal{N}_{div}(v)\}). \quad (5)$$

Here $\mathcal{N}_{sim}(v)$ and $\mathcal{N}_{div}(v)$ are the subset of neighbors of v sampled from the distribution based on similarity and diversity, respectively. We can combine these representations by concatenating, $\boldsymbol{h}_v^{(i)} = \text{MLP}(\boldsymbol{h}_{v_{sim}}^{(i)} | \boldsymbol{h}_{v_{div}}^{(i)})$ or using skip connections,

$$\boldsymbol{h}_v^{(i)} = \text{MLP}(\sigma(\boldsymbol{W}(\boldsymbol{h}_{v_{sim}}^{(i)} | \boldsymbol{h}_{v_{div}}^{(i)}) + \boldsymbol{h}_{v_{sim}}^{(i)} + \boldsymbol{h}_{v_{div}}^{(i)})). \quad (6)$$

Combining representations at the root of the computation graph as shown in Fig. 5c can be better than combining two different samples at each node of the tree, as in Fig. 5d. This avoids overfitting and makes it computationally efficient. For transductive learning,

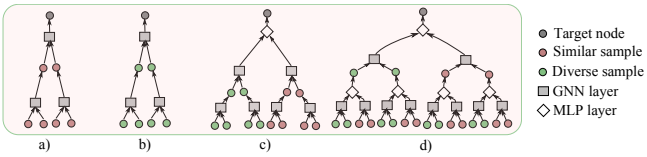


Figure 5: Computation graph with sample size $k = 2$ and hop-size 2. a), b) samples from similarity and diversity ranking for a single channel, c) dual channel with combined representation at the target node, and d) similar and diverse weighted samples at each sampled node.

as we have already precomputed the probability distributions, the inference process works similarly to the training phase. In inductive setting, we have to compute the probability distribution over the neighbors of a node as described in Section 3.1.

3.2.1 Other sampling strategies and models: AGS can also be integrated into ACM-GCN [25] and other filter-based spectral GNNs. At the start of the epoch, we sample k neighbors of each node from the similarity and diversity based distributions and construct two sparse subgraphs. We can then use the subgraph based on similarity with a low-pass filter and the subgraph based on diversity with a high-pass filter. We can employ *graph sampling* strategies (instead of node sampling), such as weighted random walks, and use them with existing GNNs. We call our graph sampling GNN `AGS-GS`.For the downstream model, there is flexibility to adapt existing models like Chebnet [13], GSAINT [46], GIN [40], GCN [18], and GAT [37]. We can also use two separate GNNs in two separate channels.

3.3 Computation Complexity

The pre-computation of the probability distribution requires $O(f|\mathcal{V}| \cdot d \log d)$ and $O(f|\mathcal{V}|d^2)$ operations for similarity and facility location based ranking, respectively. If the similarity metric is required to be learned the added time complexity is $O(f|\mathcal{E}|)$. The training and memory complexity depend on the usage of underlying GNNs. Let H, the number of hidden dimensions, be set to f for all ℓ layers. For Stochastic Gradient Descent (SGD)-based approaches, let b be the batch size, and k be the number of sampled neighbors per node. When a single channel is used, each node expands to k^ℓ nodes in its computation graph, and requires f^2 operations to compute the embedding (a matrix-vector multiplication) in every epoch. Therefore, for $|\mathcal{V}|$ nodes, the per epoch training time complexity is $O(|\mathcal{V}|k^\ell f^2)$. The memory complexity is $O(bk^\ell f + \ell f^2)$, where the first term corresponds to the space for storing the embedding, and the second term corresponds to storage for all weights of neurons of size, $\boldsymbol{W} \in \mathbb{R}^{f \times f}$.

For single channel node sampling (Fig. 5a, b), the training and memory complexity of `AGS-GNN` is similar to GSAGE. The training and memory requirements are twice as much for dual channels using the same sampling size, leaving the asymptotic bounds unchanged. However, for the dual channel with computation graph scenario shown in Fig. 5d, where each node generates two types of samples of size k, the per epoch computation complexity becomes $O(|\mathcal{V}|2^\ell k^\ell f^2)$. One way to ameliorate this cost is to reduce the sample neighborhood size by half.

4 Related Work

While *Spatial* GNNs focus on graph structure (topology) to develop aggregation strategies, *spectral* GNNs leverage graph signal processing to design graph filters. Spectral GNNs use low-pass and high-pass filters to extract low-frequency and high-frequency signals adaptively for heterophilic graphs. ACM-GCN [25] is one of the best-performing heterophilic GNNs that uses adaptive channels with low-pass and high-pass filters. Recently, the authors of [42] proposed an adaptive filter-based GNN, ALT, that combines signals from two filters with complementary filter characteristics to classify homophilic and heterophilic graphs. These methods perform well for small heterophilic graphs but do not scale to large graphs. In the *spectral* domain, AGS-GNN can be used in conjunction with these approaches by computing feature-similarity and feature-diversity-based sparse graphs at first before applying filters for large graphs.

For applying *spatial* GNNs to heterophilic graphs, rather than using average aggregation (as in homophilic GNNs), edge-aware weights of neighbors can be assigned according to the spatial graph topology and node labels. GGCN [43] uses cosine similarity to create signed neighbor features; the intuition is to send positive attributes to neighbors in the same class and negative ones to neighbors in different classes. GPNN [44] considers ranking the neighbors based on similarity and uses a 1D convolution operator on the sequential nodes. Another related work is SimP-GCN [16], which computes the node similarity and then chooses the top k similar node pairs in terms of feature-level cosine similarity for each ego node, and then constructs the neighbor set using the k-NN algorithm. AGS-GNN, in contrast, uses submodularity, node-similarity, reweighting, and sampling of the subgraph instead of reconstructing neighbor sets.

When learned weight functions are considered, AGS-GNN can be placed into the category of supervised sampling. LAGCN [2] trains an edge classifier from the existing graph topology, similar to our regression task of weight approximation. NeuralSparse [48] learns a sparsification network to sample a k-neighbor subgraph (with a predefined k), which is fed to GCN [18], GSAGE [12] or GAT [37]. Unlike our heuristic-based sampler, NeuralSparse has end-to-end training of the sampler network and GNN, and may require more iterations to find appropriate sampling probabilities.

There are only a few scalable GNNs for heterophilic graphs. The most notable one is LINKX [23], which is transductive as the model architecture depends on node size. A recent scalable GNN, LD2 [22], attempts to remedy this by transforming the adjacency and the feature matrix into embeddings as a pre-computation and then applying feature transformation in a mini-batch fashion.

5 Experiments

5.1 Dataset, Setup, and Methods

5.1.1 Dataset: We experimented with 35 graphs of different sizes and varying homophily. We also generated synthetic graphs of different homophily and degree distributions while retaining the node features for ablation studies and scaling experiments. We considered the node classification task in our experiments. For heterophily studies, we used: Cornell, Texas, Wisconsin [30]; Chameleon, Squirrel [33]; Actor [30]; Wiki, ArXiv-year, Snap-Patents, Penn94, Pokec, Genius, Twitch-Gamers, reed98, amherst41, cornell5, and Yelp [23]. We also experimented on some recent benchmark datasets, Roman-empire, Amazon-ratings, Minesweeper, Tolokers, and Questions from [32]. We converted a few multi-label multiclass classification problems (Flickr, Amazon-Products) to single-label multiclass node classification, and their homophily values became relatively small, making them heterophilic. For homophily studies we used Cora; Citeseer; pubmed; Coauthor-cs, Coauthor-physics; Amazon-computers, Amazon-photo; Reddit [12]; Reddit2 [46]; and, dblp.

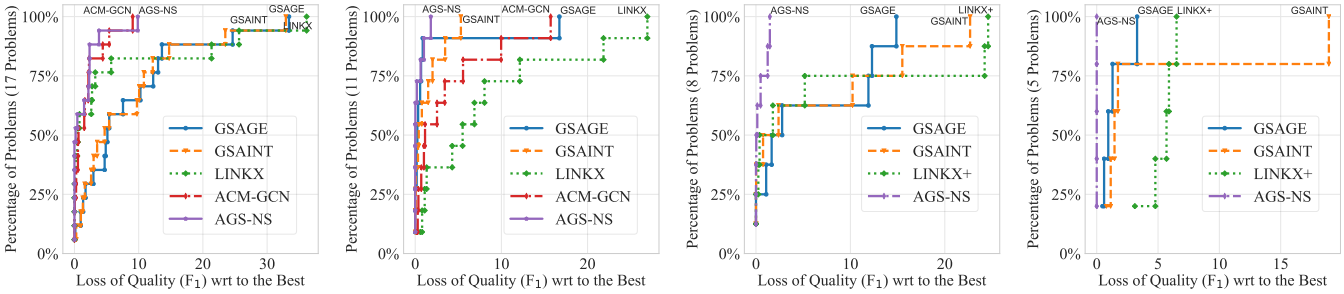
5.1.2 Experimental Setup: All evaluations are repeated ten times using a split of 60%/20%/20% (train/validation/test), unless a specific split is specified. All experiments are executed on 24GB NVIDIA A10 Tensor Core GPU. For all benchmark models, we use the settings specified in their corresponding papers, and for AGS-GNN, we use two message-passing layers ($\ell = 2$) and a hidden layer with dimension $H = 256$. We use the Adam [17] optimizer with a learning rate of 10^{-3} and train for 250 epochs or until convergence. A model converges if the standard deviation of the training loss in the most recent 5 epochs (after at least 5 epochs) is below 10^{-4}. For node sampling, we use a batch size of 512 to 1024 with neighborhood sample size $k = [25, 10]$ or $[8, 4]$ in the two hops unless otherwise specified. For graph sampling, we use a batch size of 6000 and a random walk step size of 2. For reporting accuracy, we select the model that gives the best validation performance. Depending on the models, we use a dropout probability of 0.2 or 0.5 during training. All of the implementations are in PyTorch [29], PyTorch Geometric [10], and Deep Graph Library (DGL) [38]. For computing distributions based on submodular ranking, we used the Apricot library [34]. We modified the Apricot code to make it more efficient. Source codes for all our implementations are provided on GitHub[1].

5.1.3 Implemented Methods: We consider graphs with $\mathcal{H}_a \leq 0.5$ to be heterophilic. The small instances contain graphs with less than $100K$ nodes, and they fit in the GPU memory. The larger instances are compared against only scalable homophilic and heterophilic GNNs. We compare AGS-GNN (the Node Sampling AGS-NS variant) with other Node Sampling methods (GSAGE [12]), Graph-Sampling (GSAINT [46]), and Heterophilic GNNs (LINKX [23], ACM-GCN [25]). LINKX is used only for small graphs, where the entire graph fits into GPU memory. For large graphs, we used the row-wise minibatching of the adjacency matrix (*AdjRowLoader*) for LINKX, which is denoted by LINKX+. Since, most of the heterophilic GNNs require the entire graph and do not support mini-batching, we compare AGS-GNN with 18 standard heterophilic and homophilic GNNs on small heterophilic graphs only.

5.2 Key Results

In Fig. 6, we show the performance profile plot of the algorithms w.r.t. their relative F_1-score. For each graph, we compute the relative F_1-score for all algorithms by subtracting their F_1-score from the best one. Thus, the best performing algorithm for a problem receives a score of 0, and for all other algorithms, the difference is positive. The X-axis of Fig. 6 represents these relative values from the best-performing algorithms across the graph problems, and the Y-axis shows the fraction of problems that achieve F_1-score within the bound on the difference specified by the X-axis value. Thus, the

[1]GitHub link for AGS-GNN and all other methods.



(a) Small Heterophilic Graphs **(b) Small Homophilic Graphs** **(c) Large Heterophilic Graphs** **(d) Large Homophilic Graphs**

Figure 6: *Quality Profile:* The X-axis shows the F_1 score of an algorithm relative to the best performing algorithm for problems, and the Y-axis shows the fraction of problems an algorithm solves with at most the given relative F_1 score. (See the text for details.) We compare AGS to two scalable homophilic GNNs (GSAGE, GSAINT) and two heterophilic GNNs (ACM-GCN, LINKX). For small ($|\mathcal{V}| < 100K$) and large ($|\mathcal{V}| \geq 100K$) graphs, we consider LINKX (full-batch) and LINKX+ (mini-batch), respectively. Full results are reported in Appendix 7.3.

performance plot reveals a ranking of the algorithms in terms of their quality. The closer a curve (algorithm) is to the Y-axis, and the smaller the difference in the X-axis, the better its performance w.r.t. other algorithms across all the problems considered.

Fig. 6a summarizes results from five algorithms across 17 test problems for small heterophilic graphs, where we observe that AGS-NS performs the best or close-to-best for about half of the problems and has up to 10% lower F_1 scores compared to the best algorithm for the other half. ACM-GCN performs similarly to AGS-NS for these small graphs. While LINKX achieves comparable accuracies to AGS-NS for most of the problems (about 80%), for a few problems it achieves lower F_1 scores. For large heterophilic graphs (Fig. 6c), performance improvement for AGS-NS is considerably better than all algorithms. LINKX+ performs second-best for 75% of the problems. In small homophilic graphs (Fig. 6b), AGS-NS and GSAGE are the top two performers for most of the problems, followed by GSAINT, ACM-GCN, and LINKX. This is expected since ACM-GCN and LINKX are tailored for heterophilic graphs. In large homophilic graphs (Fig. 6d), AGS-NS is the best-performing algorithm in terms of accuracy, followed by GSAGE. We also observe from this figure that for homophilic graphs, LINKX+ is not competitive.

In Table. 10 (Appendix 7.4) we present performance of AGS-NS compared to 18 recent competing algorithms on small heterophilic graphs. ACM-GCN, AGS-NS, and LINKX remain the best-performing, with AGS-NS as the leading method for these inputs. The full set of results with numerical values are provided in Appendix (Tables 6, 7, 8, 9, 10). These tables present the mean (μ) and standard deviation (σ) of the runs. We use a one-tailed statistical hypothesis t-test to verify whether one set of values is better than the other. The best results for each problem are highlighted in boldface.

5.3 Ablation study

We now investigate the contribution of individual components of AGS-GNN, employing different sampling strategies and GNNs. For GNNs, we consider GSAGE (*spatial*) and Chebnet [13] (*spectral*). We sample $\lfloor d_u \cdot k' \rfloor$ ($k' \in [0, 1]$) neighbors for each node u using *similarity* and *diversity*-based sparsification using precomputed weights (detailed in Section 3.2), and *random* sparsification that

Dataset	CoraSyn0.05				CoraSyn0.25				CoraSyn0.50			
GNN	GSAGE		Chebnet		GSAGE		Chebnet		GSAGE		Chebnet	
Sparse	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
Random	57.08	0.46	56.44	0.14	61.85	0.94	62.14	1.29	92.09	0.79	92.48	0.72
Similar	48.50	0.29	50.91	0.22	**71.66**	0.42	**71.02**	0.17	92.48	0.51	91.71	0.25
Diverse	61.79	0.17	**64.67**	0.22	53.91	0.22	57.03	0.30	79.48	0.36	76.07	0.36
Whole	**63.67**	0.43	62.79	0.25	69.49	0.38	67.61	0.22	**96.94**	0.08	**96.47**	0.14

Table 1: F_1 scores of a single sparse subgraph keeping $\lfloor 0.25 \cdot d_u \rfloor$ neighbors of node u. Three synthetic versions of Cora are produced ($d = 200$ and $\mathcal{H}_n$ 0.05, 0.25, and 0.50, respectively).

Dataset	CoraSyn0.05				CoraSyn0.25				CoraSyn0.50			
GNN	GSAGE		Chebnet		GSAGE		Chebnet		GSAGE		Chebnet	
Sampler	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
Random	59.20	0.22	57.50	0.80	62.85	0.55	61.67	0.46	91.06	0.36	88.89	1.32
Similar	54.44	0.68	54.03	0.51	**69.19**	0.08	**67.43**	0.30	**92.00**	0.08	**91.71**	0.29
Diverse	**60.96**	0.55	**61.73**	0.43	56.67	0.46	58.38	0.25	86.65	0.44	83.54	0.79

Table 2: F_1 scores using different subgraph samples ($k = [25, 25]$). Three synthetic versions of Cora are produced ($d = 200$ and $\mathcal{H}_n$ 0.05, 0.25, and 0.50, respectively).

selects a subgraph uniformly at random. Only a *single* sampled subgraph is used throughout the training. For experiments, we generate three synthetic versions of Cora with average degree $d = 200$, keeping the original nodes and features the same but changing the connectivity to have strong (0.05), moderate (0.25), and weak (0.50) heterophily. The distribution of heterophily for each node is close to uniform. Table 1 shows that diversity-based selection performs best with strong heterophily, and with spectral GNN, it even achieves accuracy better than using the entire graph. In contrast, on moderate heterophily, the similarity-based selection performs the best (even better than the whole graph). For weak heterophily (homophily), similar and random sparse perform alike. When we convert these into a sampling paradigm (node sampling or graph sampling), similar behavior can be seen (Table 2) as we get the best performance from diversity-based selection for strong heterophily and similarity-based for moderate heterophily with a wide margin over random. For weak heterophily or homophily, similarity-based sampling performs slightly better than random.

ACM-GCN		LINKX		AGS (Sim. + Sim.)		AGS (Div. + Div.)		AGS (Sim. + Div.)	
μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
35.33	0.69	34.20	0.78	60.81	0.77	62.83	0.85	**63.03**	0.50

Table 3: F_1 scores of two-channel GNNs with different weighted samplers in a synthetic Cora graph with a mix of locally heterophilic and homophilic nodes ($\mathcal{H}_n(u) = [0.05, 0.50]$).

Since real-world graphs have nonuniform node homophily, we generated a synthetic version of `Cora` where individual nodes have different local node homophily in the range $[0.05, 0.50]$. Table 3 shows that `AGS-GNN`with dual channels (one for homophily and one for heterophily) performs the best, significantly outperforming ACM-GCN and LINKX.

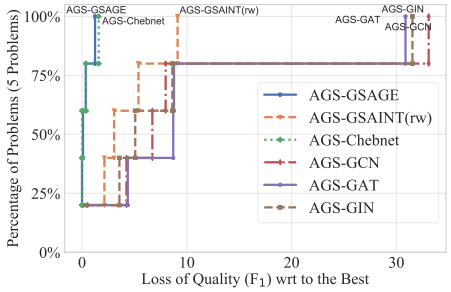


Figure 7: *Quality Profile:* X-axis is the relative F_1 scores for different GNNs coupled with the AGS sampler on five benchmark heterophilic graphs. The Y-axis shows the fraction of problems solved with at most a given relative F_1 score.

We coupled our node and graph sampling strategies to existing GNNs (GSAGE, Chebnet, GSAINT, GIN, GAT, and GCN) and evaluated them on five heterophilic graphs (`Reed98, Roman-empire, Actor, Minesweeper, Tolokers`). We summarize the key results as a performance profile in Fig. 7 and observe that AGS with GSAGE, Chebnet, and GSAINT performed the best.

5.4 Experimental Runtime and Convergence

Table 4 shows per epoch training time and Table 5 shows pre-computation time of different methods under the same settings. For large datasets such as `Reddit`, with a single worker thread our current implementation of weighted sampling requires about 3 times more time than the random sampling used in GSAGE due to the dual channels and a few implementation differences with PyTorchGeometric. The pre-computation time for similarity ranking is faster than that for submodular-based diversity-based ranking. A faster computation of diversity ranking is left for future work. Fig. 8 shows the number of epochs required to converge for random sampling (GSAGE) and our weighted sampling (`AGS-GNN`). Using the same settings, we see that `AGS-GNN` is more stable and requires fewer epochs on average to converge than GSAGE.

6 Conclusions

In this work, we proposed attribute-guided sampling that uses node features in an unsupervised and supervised fashion. We have shown that through a biased sampling of similar and diverse neighbors, we get improved performance in homophilic graphs and can handle challenging heterophilic graphs. We verify our claims through exhaustive experimentation on various benchmark datasets and methods. A limitation of our work is the time required for submodular optimization when the facility location function is used; the computation complexity is higher for dense graphs. We will optimize this implementation in our future work and build an end-to-end process for supervised sampling.

Method	Settings	Reddit	Genius	Yelp
GSAGE	k = [25, 10], b = 1024, ℓ = 2, H=256	5.47	2.16	7.71
GSAINT	b = 4096, rw, step = 2, ℓ = 2, H = 256	8.01	2.10	4.34
LINKX+	random 100 parts, b = 2, H = 32	10.63	2.35	3.54
AGS-NS	k = [25, 10], b = 1024, ℓ = 2, H = 256	18.44	2.43	13.30
AGS-GS	b = 4096, wrw, step = 2, ℓ = 2, H = 256	5.90	1.72	4.06

Table 4: Average training time (seconds/epoch). `rw` and `wrw` refer to random walk and weighted random walk, respectively.

Method	Settings	Reddit	Genius	Yelp
GSAINT	Norm Computation	414.93	92.30	215.99
Sim. Ranking	step, k_1, k_2 = 20%	60.03	6.00	41.20
Div. Ranking	step, k_1, k_2 = 20%	600.00	16.37	55.62
Learning Sim. (epoch)	H = 256, b = 10000	7.01	1.00	4.03

Table 5: Pre-computation time (seconds) of different components of AGS-GNN and GSAINT with a single worker thread. For similarity and diversity ranking, 'step' is used as a probability mass function where k_1 and k_2 percentage of neighbors are assigned the same weight.

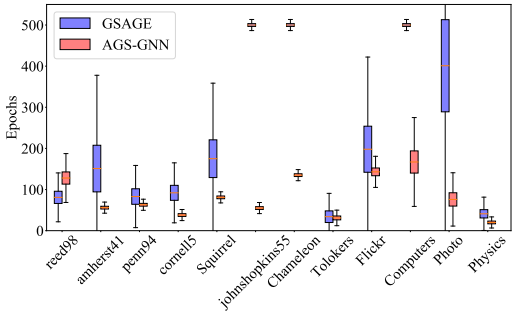


Figure 8: Number of epochs for `AGS-GNN` and GSAGE to converge. Here, methods are run to a maximum epoch of 500.

Acknowledgments

This work is supported at the Pacific Northwest National Laboratory by the U.S. Department of Energy through the Exascale Computing Project (17-SC-20-SC) (ExaGraph), and the Laboratory Directed Research and Development (LDRD) Program. At Purdue, work is supported by ExaGraph and the Advanced Scientific Computing Research program of the Office of Science through grant DE-SC0022260. At Boise State University, work is supported by the National Centers of Academic Excellence in Cybersecurity grant (H98230-22-1-0300), which is part of the National Security Agency.

References

[1] Sami Abu-El-Haija, Bryan Perozzi, Amol Kapoor, Nazanin Alipourfard, Kristina Lerman, et al. 2019. MixHop: Higher-Order Graph Convolutional Architectures via Sparsified Neighborhood Mixing. In *The 36th International Conference on Machine Learning*, Vol. 97. PMLR, Long Beach, California, USA, 21–29.

[2] Hao Chen, Yue Xu, Feiran Huang, Zengde Deng, Wenbing Huang, Senzhang Wang, Peng He, and Zhoujun Li. 2020. Label-Aware Graph Convolutional Networks. In *The 29th International Conference on Information and Knowledge Management*. ACM, Virtual Event, Ireland, 1977–1980.

[3] Jie Chen, Tengfei Ma, and Cao Xiao. 2018. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In *The 6th International Conference on Learning Representations*. OpenReview.net, Vancouver, BC, Canada.

[4] Jianfei Chen, Jun Zhu, and Le Song. 2018. Stochastic Training of Graph Convolutional Networks with Variance Reduction. In *The 35th International Conference on Machine Learning*, Jennifer G. Dy and Andreas Krause (Eds.), Vol. 80. PMLR, Stockholm, Sweden, July, 941–949.

[5] Ming Chen, Zhewei Wei, Zengfeng Huang, Bolin Ding, and Yaliang Li. 2020. Simple and Deep Graph Convolutional Networks. In *The 37th International Conference on Machine Learning*, Vol. 119. PMLR, Virtual Event, 1725–1735.

[6] Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. 2019. Cluster-GCN: An Efficient Algorithm for Training Deep and Large Graph Convolutional Networks. In *The 25th International Conference on Knowledge Discovery & Data Mining, KDD*. ACM, Anchorage, AK, USA, 257–266.

[7] Davide Chicco. 2021. Siamese Neural Networks: An Overview. In *Artificial Neural Networks - Third Edition*. Vol. 2190. Springer, 73–94.

[8] Eli Chien, Jianhao Peng, Pan Li, and Olgica Milenkovic. 2021. Adaptive Universal Generalized PageRank Graph Neural Network. In *The 9th International Conference on Learning Representations*. OpenReview.net, Virtual Event, Austria.

[9] Venkatesan Nallampatti Ekambaram. 2014. *Graph Structured Data Viewed Through a Fourier Lens*. Ph. D. Dissertation. http://www.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-209.html

[10] Matthias Fey and Jan Eric Lenssen. 2019. Fast Graph Representation Learning with PyTorch Geometric. *CoRR* abs/1903.02428 (2019).

[11] Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville. 2016. *Deep Learning*. MIT Press.

[12] William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *The Annual Conference on Neural Information Processing Systems 2017*. Long Beach, CA, USA, 1024–1034.

[13] Mingguo He, Zhewei Wei, and Ji-Rong Wen. 2022. Convolutional neural networks on graphs with chebyshev approximation, revisited. *Advances in Neural Information Processing Systems* 35 (2022), 7264–7276.

[14] Qian Huang, Horace He, Abhay Singh, Ser-Nam Lim, and Austin R. Benson. 2021. Combining Label Propagation and Simple Models out-performs Graph Neural Networks. In *The 9th International Conference on Learning Representations*. OpenReview.net, Virtual Event, Austria.

[15] Wen-bing Huang, Tong Zhang, Yu Rong, and Junzhou Huang. 2018. Adaptive Sampling Towards Fast Graph Representation Learning. In *The Annual Conference on Neural Information Processing Systems*. Montréal, Canada, 4563–4572.

[16] Wei Jin, Tyler Derr, Yiqi Wang, Yao Ma, Zitao Liu, and Jiliang Tang. 2021. Node Similarity Preserving Graph Convolutional Networks. In *The Fourteenth ACM International Conference on Web Search and Data Mining*. ACM, Virtual Event, Israel, 148–156.

[17] Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *The 3rd International Conference on Learning Representations*. OpenReview.net, San Diego, CA, USA.

[18] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *The 5th International Conference on Learning Representations*. OpenReview.net, Toulon, France.

[19] Johannes Klicpera, Aleksandar Bojchevski, and Stephan Günnemann. 2019. Predict then Propagate: Graph Neural Networks meet Personalized PageRank. In *The 7th International Conference on Learning Representations*. OpenReview.net, New Orleans, LA, USA.

[20] Andreas Krause and Daniel Golovin. 2014. Submodular Function Maximization. In *Tractability: Practical Approaches to Hard Problems*. Vol. 3. Cambridge University Press, 71–104.

[21] Xiang Li, Renyu Zhu, Yao Cheng, Caihua Shan, Siqiang Luo, Dongsheng Li, and Weining Qian. 2022. Finding Global Homophily in Graph Neural Networks When Meeting Heterophily. In *The International Conference on Machine Learning*, Vol. 162. PMLR, Baltimore, Maryland, USA, 13242–13256.

[22] Ningyi Liao, Siqiang Luo, Xiang Li, and Jieming Shi. 2023. LD2: Scalable Heterophilous Graph Neural Network with Decoupled Embeddings. In *The Annual Conference on Neural Information Processing Systems*. New Orleans, LA, USA.

[23] Derek Lim, Felix Hohne, Xiuyu Li, Sijia Linda Huang, Vaishnavi Gupta, Omkar Bhalerao, and Ser Nam Lim. 2021. Large Scale Learning on Non-Homophilous Graphs: New Benchmarks and Strong Simple Methods. *Advances in Neural Information Processing Systems* 34 (2021), 20887–20902.

[24] Meng Liu, Zhengyang Wang, and Shuiwang Ji. 2021. Non-local graph neural networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, 12 (2021), 10270–10276.

[25] Sitao Luan, Chenqing Hua, Qincheng Lu, Jiaqi Zhu, Mingde Zhao, Shuyuan Zhang, Xiao-Wen Chang, and Doina Precup. 2022. Revisiting Heterophily For Graph Neural Networks. In *The Annual Conference on Neural Information Processing Systems*. New Orleans, LA, USA.

[26] Sitao Luan, Mingde Zhao, Chenqing Hua, Xiao-Wen Chang, and Doina Precup. 2022. Complete the Missing Half: Augmenting Aggregation Filtering with Diversification for Graph Convolutional Neural Networks. *CoRR* abs/2212.10822 (2022).

[27] Michel Minoux. 1978. Accelerated Greedy Algorithms for Maximizing Submodular Set Functions. *Optimization Techniques. Lecture Notes in Control and Information Sciences* 7 (1978), 234–243.

[28] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. 1978. An Analysis of Approximations for Maximizing Submodular Set Functions—I. *Mathematical Programming* 14 (1978), 265–294.

[29] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *The Annual Conference on Neural Information Processing Systems*, Vol. 32. Vancouver, BC, Canada, 8024–8035.

[30] Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. 2020. Geom-GCN: Geometric Graph Convolutional Networks. In *The 8th International Conference on Learning Representations*. OpenReview.net, Addis Ababa, Ethiopia.

[31] Oleg Platonov, Denis Kuznedelev, Artem Babenko, and Liudmila Prokhorenkova. 2023. Characterizing Graph Datasets for Node Classification: Homophily-Heterophily Dichotomy and Beyond. In *The Annual Conference on Neural Information Processing Systems 2023*. New Orleans, LA, USA.

[32] Oleg Platonov, Denis Kuznedelev, Michael Diskin, Artem Babenko, and Liudmila Prokhorenkova. 2023. A Critical Look at the Evaluation of GNNs under Heterophily: Are We Really Making Progress?. In *The 11th International Conference on Learning Representations*. OpenReview.net, Kigali, Rwanda.

[33] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. 2021. Multi-scale Attributed Node Embedding. *Journal of Complex Networks* 9, 2 (2021), cnab014.

[34] Jacob Schreiber, Jeffrey Bilmes, and William Stafford Noble. 2020. Apricot: Submodular Selection for Data Summarization in Python. *The Journal of Machine Learning Research* 21, 1 (2020), 6474–6479.

[35] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Gallagher, and Tina Eliassi-Rad. 2008. Collective Classification in Network Data. *AI Magazine* 29, 3 (2008), 93–93.

[36] Jesper E Van Engelen and Holger H Hoos. 2020. A survey on semi-supervised learning. *Machine Learning* 109, 2 (2020), 373–440.

[37] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *The 6th International Conference on Learning Representations*. OpenReview.net, Vancouver, BC, Canada.

[38] Minjie Wang, Lingfan Yu, Da Zheng, Quan Gan, Yu Gai, et al. 2019. Deep Graph Library: Towards Efficient and Scalable Deep Learning on Graphs. *CoRR* abs/1909.01315 (2019).

[39] Felix Wu, Amauri H. Souza Jr., Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Q. Weinberger. 2019. Simplifying Graph Convolutional Networks. In *The 36th International Conference on Machine Learning*, Vol. 97. PMLR, Long Beach, California, USA, 6861–6871.

[40] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *The 7th International Conference on Learning Representations*. OpenReview.net, New Orleans, LA, USA.

[41] Keyulu Xu, Chengtao Li, Yonglong Tian, Tomohiro Sonobe, Ken-ichi Kawarabayashi, and Stefanie Jegelka. 2018. Representation Learning on Graphs with Jumping Knowledge Networks. In *The 35th International Conference on Machine Learning*, Vol. 80. PMLR, Stockholm, Sweden, 5449–5458.

[42] Zhe Xu, Yuzhong Chen, Qinghai Zhou, et al. 2023. Node Classification Beyond Homophily: Towards a General Solution. In *The 29th Conference on Knowledge Discovery and Data Mining*. ACM, Long Beach, CA, USA, 2862–2873.

[43] Yujun Yan, Milad Hashemi, Kevin Swersky, Yaoqing Yang, and Danai Koutra. 2022. Two Sides of the Same Coin: Heterophily and Oversmoothing in Graph Convolutional Neural Networks. In *International Conference on Data Mining*, Xingquan Zhu, Sanjay Ranka, My T. Thai, Takashi Washio, and Xindong Wu (Eds.). IEEE, Orlando,FL, USA, 1287–1292.

[44] Tianmeng Yang, Yujing Wang, Zhihan Yue, Yaming Yang, Yunhai Tong, and Jing Bai. 2022. Graph Pointer Neural Networks. In *The 36th AAAI Conference on Artificial Intelligence*. AAAI Press, Virtual Event, 8832–8839.

[45] Yang Ye and Shihao Ji. 2019. Sparse Graph Attention Networks. *CoRR* abs/1912.00552 (2019).

[46] Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor K. Prasanna. 2020. GraphSAINT: Graph Sampling Based Inductive Learning Method. In *The 8th International Conference on Learning Representations*. OpenReview.net, Addis Ababa, Ethiopia.

[47] Elena Zheleva and Lise Getoor. 2009. To Join or Not to Join: The Illusion of Privacy in Social Networks with Mixed Public and Private User Profiles. In *The*

18th International Conference on World Wide Web. ACM, Madrid, Spain, 531–540.
[48] Cheng Zheng, Bo Zong, Wei Cheng, Dongjin Song, et al. 2020. Robust Graph Representation Learning via Neural Sparsification. In *The 37th International Conference on Machine Learning*, Vol. 119. PMLR, Virtual Event, 11458–11468.
[49] Xin Zheng, Yixin Liu, Shirui Pan, Miao Zhang, Di Jin, and Philip S. Yu. 2022. Graph Neural Networks for Graphs with Heterophily: A Survey. *CoRR* abs/2202.07082 (2022).
[50] Jiong Zhu, Ryan A. Rossi, Anup Rao, et al. 2021. Graph Neural Networks with Heterophily. In *The 35th AAAI Conference on Artificial Intelligence*. AAAI Press, Virtual Event, 11168–11176.
[51] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. 2020. Beyond Homophily in Graph Neural Networks: Current Limitations and Effective Designs. *Advances in Neural Information Processing Systems* 33 (2020), 7793–7804.

7 Appendix

7.1 Proof of Lemma 2.1: Homophily of similarity-based selection

Proof. Consider an ego node $t = \{\boldsymbol{x}_t, y_t\}$ where $\boldsymbol{x}_t$, and y_t are its feature and label, respectively. Let the feature and labels of the neighboring nodes of t be $\mathcal{N}(t) = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_{d_t}, y_{d_t})\}$.

Let $s(\boldsymbol{x}_i, \boldsymbol{x}_t)$ be a similarity function that measures how similar the feature of a neighbor x_i is to the feature of the ego node $\boldsymbol{x}_t$. This function returns a positive value, and higher values indicate higher similarity. A probability distribution with probability mass function $p_{\mathcal{S}}(i)$ assigns a probability to each neighbor i based on its similarity to the ego node. The distribution should satisfy the following properties:

- $p_{\mathcal{S}}(i) \geq 0$ for all i; $\sum_{i=1}^{d_t} p_{\mathcal{S}}(i) = 1$;
- $p_{\mathcal{S}}(i)$ is proportional to $s(\boldsymbol{x}_i, \boldsymbol{x}_t)$. (e.g., $p_{\mathcal{S}}(i) = \frac{s(\boldsymbol{x}_i, \boldsymbol{x}_t)}{\sum_{j=1}^{d_t} s(\boldsymbol{x}_j, \boldsymbol{x}_t)}$.)

Let $I(y_i = y_t)$ be an indicator function that returns 1 if $y_i = y_t$ and 0 otherwise. If a neighbor i of a node t is selected randomly then $p_{\mathcal{U}}(i) = \frac{1}{d_t}$. Therefore the probability of selecting a neighbor randomly having the same label as the ego node is,

$$P_{\mathcal{U}}(y_i = y_t) = \sum_{i=1}^{d_t} p_{\mathcal{U}}(i) \cdot I(y_i = y_t) = \frac{1}{d_t} \cdot \sum_{i=1}^{d_t} I(y_i = y_t).$$

If the selection probability is based on similarity, then

$$\begin{aligned} P_{\mathcal{S}}(y_i = y_t) &= \sum_{i=1}^{d_t} p_{\mathcal{S}}(i) \cdot I(y_i = y_t) \\ &= \sum_{i=1}^{d_t} \frac{s(\boldsymbol{x}_i, \boldsymbol{x}_t)}{\sum_{j=1}^{d_t} s(\boldsymbol{x}_j, \boldsymbol{x}_t)} \cdot I(y_i = y_t). \end{aligned}$$

Let n be the number of neighbors having the same label as ego node t, i.e., $n = \sum_{i=1}^{d_t} I(y_i = y_t)$. Let s_t be the sum of similarities of all neighbors having the same label as t, i.e., $s_t = \sum_{i=1}^{d_t} (s(\boldsymbol{x}_i, \boldsymbol{x}_t) \cdot I(y_i = y_t)$. Also let s_d denote the sum of similarities of all neighbors of t, then $s_d = \sum_{j=1}^{d_t} s(\boldsymbol{x}_j, \boldsymbol{x}_t)$.

We can now use these expressions and the substitutions shown below to derive the result. From Assumption 2.1,

$$\frac{s_t}{n} \geq \frac{s_d}{d_t} \implies \frac{s_t}{s_d} \geq \frac{n}{d_t},$$

$$\frac{1}{\sum_{j=1}^{d_t} s(x_j, x_t)} \sum_{i=1}^{d_t} s(\boldsymbol{x}_i, \boldsymbol{x}_t) \cdot I(y_i = y_t) \geq \frac{1}{d_t} \cdot \sum_{i=1}^{d_t} I(y_i = y_t),$$

$$\sum_{i=1}^{d_t} \frac{s(\boldsymbol{x}_i, \boldsymbol{x}_t)}{\sum_{j=1}^{d_t} s(\boldsymbol{x}_j, \boldsymbol{x}_t)} \cdot I(y_i = y_t) \geq \sum_{i=1}^{d_t} \frac{1}{d_t} \cdot I(y_i = y_t),$$

$$\sum_{i=1}^{d_t} p_{\mathcal{S}}(i) \cdot I(y_i = y_t) \geq \sum_{i=1}^{d_t} p_{\mathcal{U}}(i) \cdot I(y_i = y_t),$$

$$P_{\mathcal{S}}(y_i = y_t) \geq P_{\mathcal{U}}(y_i = y_t),$$

$$\mathcal{H}'_n(t) \geq \mathcal{H}_n(t).$$

□

7.2 Proof of Lemma 2.2: Homophily of diversity-based selection

Proof. Let the probability mass function of the distribution be proportional to the marginal gain values of the greedy submodular algorithm (Algorithm 3, i.e., $p_{\mathcal{D}}(i) = f_i / \sum_{j=1}^{d_t} f_j$. The higher the marginal gain, the higher the selection probability.

If a neighbor is selected randomly, then $p_{\mathcal{U}}(i) = \frac{1}{d_t}$. Therefore, the probability of selecting a neighbor randomly having the same label as the ego node is

$$P_{\mathcal{U}}(y_i = y_t) = \frac{1}{d_t} \cdot \sum_{i=1}^{d_t} I(y_i = y_t).$$

If the selection probability is based on the marginal gain, then

$$\begin{aligned} P_{\mathcal{D}}(y_i = y_t) &= \sum_{i=1}^{d_t} p_{\mathcal{D}}(i) \cdot I(y_i = y_t) \\ &= \sum_{i=1}^{d_t} \frac{f_i}{\sum_{j=1}^{d_t} f_j} \cdot I(y_i = y_t). \end{aligned}$$

Let n be the number of neighbors having the same label as the ego node t, i.e., $n = \sum_{i=1}^{d} I(y_i = y_t)$. Let m_t be the sum of marginal gains of all neighbors having the same label as t, i.e., $m_t = \sum_{i=1}^{d_t} (f_i) \cdot I(y_i = y_t)$, and let m_d denote the sum of all marginal gains of the neighbors $m_d = \sum_{j=1}^{d_t} f_j$.

We use the above expressions, and the substitutions shown below get the desired result. From Assumption 2.2,

$$\frac{m_t}{n} \leq \frac{m_d}{d_t} \implies \frac{m_t}{m_d} \leq \frac{n}{d_t},$$

$$\frac{1}{\sum_{j=1}^{d} f_j} \sum_{i=1}^{d_t} f_i \cdot I(y_i = y_t) \leq \frac{1}{d_t} \cdot \sum_{i=1}^{d} I(y_i = y_t),$$

$$\sum_{i=1}^{d_t} \frac{f_i}{\sum_{j=1}^{d_t} f_j} \cdot I(y_i = y_t) \leq \sum_{i=1}^{d_t} \frac{1}{d_t} \cdot I(y_i = y_t),$$

$$\sum_{i=1}^{d_t} p_{\mathcal{D}}(i) \cdot I(y_i = y_t) \leq \sum_{i=1}^{d_t} p_{\mathcal{U}}(i) \cdot I(y_i = y_t),$$

$$P_{\mathcal{D}}(y_i = y_t) \leq P_{\mathcal{U}}(y_i = y_t),$$

$$\mathcal{H}'_n(t) \leq \mathcal{H}_n(t).$$

□

7.3 AGS-GNN performance comparison

Table 6,7,8,9 shows numerical results of the algorithms for small and large heterophilic and homophilic graphs.

Small Heterophilic	GSAGE		GSAINT		LINKX†		ACMGCN		AGS-NS	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
Cornell	71.35	6.07	67.03	3.15	**76.76**	4.32	74.59	1.32	74.59	2.16
Texas	77.30	5.57	79.46	6.53	81.62	2.02	84.32	7.13	**84.86**	6.19
Wisconsin	79.61	3.64	79.61	6.86	83.53	4.74	**84.31**	3.28	81.96	4.71
reed98	61.87	0.53	64.15	0.69	66.63	1.37	66.11	1.25	**66.74**	1.37
amherst41	66.62	0.33	69.57	0.71	78.64	0.35	78.12	0.30	**79.19**	0.47
penn94	75.65	0.42	75.11	0.33	**85.92**	0.32	85.38	0.53	76.06	0.41
Roman-empire	79.52	0.42	77.51	0.47	59.14	0.45	71.42	0.39	**80.49**	0.48
cornell5	69.22	0.12	68.10	0.15	80.10	0.27	78.43	0.50	**82.84**	0.01
Squirrel	38.66	1.24	39.14	1.45	35.91	1.09	**72.06**	2.21	68.24	0.97
johnshopkins55	67.37	0.54	67.43	0.20	**79.63**	0.16	77.37	0.61	78.13	0.42
Actor	34.82	0.55	35.24	0.81	33.93	0.82	34.42	1.08	**36.55**	0.93
Minesweeper	**85.74**	0.25	85.46	0.49	80.02	0.03	80.33	0.23	85.56	0.28
Questions	97.13	0.01	97.18	0.04	97.06	0.03	97.02	0.00	**97.27**	0.04
Chameleon	51.18	2.70	52.32	2.47	50.18	2.01	**75.81**	1.67	73.46	2.29
Tolokers	79.15	0.32	78.89	0.37	80.07	0.53	80.45	0.54	**80.52**	0.41
Flickr	50.86	0.32	50.28	0.11	**53.81**	0.31	52.19	0.24	51.52	0.13
Amazon-ratings	48.08	0.38	52.21	0.27	52.68	0.26	52.94	0.23	**53.21**	0.46

Table 6: Micro F_1-measure in small heterophilic graphs ($|\mathcal{V}| < 100K$).

Small Homophilic	GSAGE		GSAINT		LINKX†		ACMGCN		AGS-NS	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
cora	52.59	0.25	65.85	0.27	57.18	0.16	66.79	0.24	**69.32**	0.11
CiteSeer	**71.14**	0.10	65.84	3.86	44.20	2.49	55.42	3.43	69.33	0.31
dblp	85.80	0.16	85.65	0.21	80.47	0.04	85.68	0.24	**85.97**	0.11
Computers	91.31	0.11	91.77	0.14	91.38	0.17	92.03	0.48	**92.18**	0.08
pubmed	89.00	0.09	88.57	0.14	85.05	0.25	83.79	0.15	**89.34**	0.07
cora_ml	**88.71**	0.23	87.21	0.37	80.67	0.32	85.28	0.12	87.70	0.08
Cora	80.56	0.52	79.12	1.22	59.28	3.80	71.16	0.98	**81.13**	0.90
CS	95.13	0.19	**95.81**	0.07	94.48	0.12	94.80	0.34	95.18	0.06
Photo	**96.58**	0.09	96.51	0.10	95.48	0.15	96.21	0.11	96.56	0.16
Physics	96.64	0.05	**96.81**	0.09	96.23	0.05	96.11	0.24	96.62	0.04
citeseer	95.15	0.20	95.32	0.16	88.53	0.27	94.26	0.19	**95.40**	0.10

Table 7: Micro F_1-measure in small homophilic graphs ($|\mathcal{V}| < 100K$).

Large Heterophilic	GSAGE		GSAINT		LINKX		AGS-NS	
	μ	σ	μ	σ	μ	σ	μ	σ
genius	81.76	0.33	82.09	0.19	82.59	0.02	**82.84**	0.01
pokec	68.91	0.03	68.18	0.12	**70.57**	0.3	70.08	0.00
arxiv-year	47.5	0.15	40.04	0.32	49.89	0.18	**50.27**	0.10
snap-patents	**48.36**	0.01	32.86	0.11	43.19	1.86	**48.22**	0.02
twitch-gamer	**61.41**	0.00	61.39	0.32	59.62	0.18	61.38	0.05
AmazonProducts	62.96	0.00	**75.25**	0.05	50.66	0.28	73.78	0.01
Yelp	65.15	0.00	**77.06**	0.07	52.84	2.4	75.82	0.01

Table 8: Micro F_1-measure in large heterophilic graphs ($|\mathcal{V}| \geq 100K$).

Large Homophilic	GSAGE		GSAINT		LINKX		AGS-NS	
	μ	σ	μ	σ	μ	σ	μ	σ
Reddit	95.17	0.01	93.91	0.03	92.53	0.26	**95.64**	0.06
Reddit2	88.94	0.32	73.3	2.2	86.54	0.44	**92.23**	0.11
Reddit0.525	91.31	0.06	90.46	0.05	87.13	0.18	**91.91**	0.06
Reddit0.425	89.33	0.07	89.13	0.06	84.38	0.61	**90.27**	0.1
Reddit0.325	87.19	0.12	87.69	0.15	81.97	0.31	**88.48**	0.05

Table 9: Micro F_1-measure in large homophilic graphs ($|\mathcal{V}| \geq 100K$).

7.4 AGS-GNN vs. other homophilic and heterophilic GNNs

Table. 10 shows the performance of AGS relative to the 18 recent algorithms for small heterophilic graphs. We can see that ACM-GCN, AGS-NS, and LINKX are the best-performing, with AGS-NS the best among them.

GNNs	reed98		amherst41		Roman-empire		cornell5		Actor	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
GSAGE [12]	61.87	0.53	66.62	0.33	79.52	0.42	69.22	0.12	34.82	0.55
GCN [18]	60.83	0.53	57.23	1.91	42.78	1.02	57.62	0.11	29.18	0.78
GAT [37]	59.59	0.46	57.09	0.54	41.73	0.73	57.64	0.5	30.64	0.8
GIN [40]	60.1	0.87	59.82	1.45	47.34	0.26	58.74	0.18	28.09	0.25
GSAINT [46]	64.15	0.69	69.57	0.71	77.51	0.47	68.1	0.15	35.24	0.81
LINKX [23]	66.63	1.37	79.64	0.35	59.14	0.45	80.1	0.27	33.93	0.82
ACM-GCN [25]	66.11	1.25	78.12	0.3	71.42	0.39	78.43	0.5	34.42	1.08
LINK [47]	60.62	2.84	70.11	1.31	8.59	0.5	71.49	0.44	22.54	1.54
MLP [11]	41.14	0.9	51.81	2.75	65.31	0.32	61.12	1.45	33.91	0.7
C&S [14]	44.66	3.23	49.71	3.39	65.42	0.3	60.12	0.96	32.41	0.68
SGC [39]	58.86	2.38	68.86	2.48	42.32	0.63	69.14	0.66	27.91	0.71
GPRGNN [8]	50.67	3.95	58.08	1.92	69.68	0.3	63.75	0.96	29.49	0.97
APPNP [19]	53.37	1.99	62.55	2.22	56.99	0.26	66.78	0.79	25.75	0.4
MixHop [1]	57.41	2.61	68.99	2.07	78.49	0.24	70.09	0.98	32.21	0.86
GCNJK [41]	58.34	3.01	71.41	2.74	58.32	0.46	68.85	0.77	26.14	0.68
GATJK [41]	61.35	4.23	70.29	1.11	71.54	0.67	69.7	0.31	26.42	0.91
LINKConcat [47]	59.79	0.96	76.06	1.47	12.06	1.61	76.87	0.33	28.09	1.65
GCNII [5]	57.51	2.77	70.32	1.48	74.82	0.26	71.97	0.28	26.29	0.27
AGS-NS	**66.74**	1.37	**79.19**	0.47	**80.49**	0.48	**82.84**	0.01	**36.55**	0.93

Table 10: F_1 scores of additional GNNs (both homophilic and heterophilic) on small heterophilic graphs ($|\mathcal{V}| < 100K$).

7.5 AGS with existing GNNs

Table 11 compares the performance of AGS with existing GNNs (GSAGE, Chebnet, GSAINT, GIN, GAT, and GCN) evaluated on the heterophilic graphs (`Reed98`, `Roman-empire`, `Actor`, `Minesweeper`, `Tolokers`).

Methods / Dataset	AGS-GSAGE		AGS-GSAINT		AGS-Chebnet		AGS-GCN		AGS-GAT		AGS-GIN	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
reed98	63.32	0.76	55.44	3.04	**64.56**	0.84	56.58	1.20	55.85	1.33	59.48	2.00
Roman-empire	77.35	0.43	74.39	0.41	**77.44**	0.24	44.36	1.76	46.57	0.75	45.91	0.26
Actor	**35.36**	0.49	29.97	0.52	33.76	0.73	28.64	0.75	26.58	1.01	26.74	0.30
Minesweeper	**84.45**	0.41	82.31	0.68	**84.45**	0.62	80.21	0.08	80.08	0.16	80.88	0.13
Tolokers	78.33	0.14	**78.69**	0.39	78.31	0.35	78.16	0.00	78.16	0.00	78.38	0.11

Table 11: F_1 scores of heterophilic graphs using AGS sampler with different underlying GNNs. The best-performing models are AGS-GSAGE and AGS-Chebnet.