GraphReach: Position-Aware Graph Neural Network using Reachability Estimations

Sunil Nishad[1] , **Shubhangi Agarwal**[1] , **Arnab Bhattacharya**[1] and **Sayan Ranu**[2]
[1]Indian Institute of Technology Kanpur, India
[2]Indian Institute of Technology Delhi, India
snishad@cse.iitk.ac.in, sagarwal@cse.iitk.ac.in, arnabb@cse.iitk.ac.in, sayanranu@cse.iitd.ac.in

Abstract

Majority of the existing graph neural networks (GNN) learn node embeddings that encode their local neighborhoods but not their positions. Consequently, two nodes that are vastly distant but located in similar local neighborhoods map to similar embeddings in those networks. This limitation prevents accurate performance in predictive tasks that rely on position information. In this paper, we develop GRAPHREACH, a *position-aware* inductive GNN that captures the global positions of nodes through *reachability estimations* with respect to a set of *anchor* nodes. The anchors are strategically selected so that reachability estimations across all the nodes are maximized. We show that this combinatorial anchor selection problem is NP-hard and, consequently, develop a greedy $(1-1/e)$ approximation heuristic. Empirical evaluation against state-of-the-art GNN architectures reveal that GRAPHREACH provides up to 40% relative improvement in accuracy. In addition, it is more robust to adversarial attacks.

1 Introduction and Related Work

Learning feature-space node embeddings through graph neural networks (GNN) has received much success in tasks such as node classification, link prediction, graph generation, learning combinatorial algorithms, etc. [Hamilton *et al.*, 2017; Kipf and Welling, 2017; Velickovic *et al.*, 2018; Goyal *et al.*, 2020; Manchanda *et al.*, 2020]. GNNs learn node embeddings by first collecting structural and attribute information from the neighborhood of a node and then encoding this information into a feature vector via non-linear transformation and aggregation functions. This allows GNNs to generalize to unseen nodes, i.e., nodes that were not present during the training phase. This *inductive* learning capability of GNNs is one of the key reasons behind their popularity.

Due to reliance on only the neighborhood information, most GNN architectures fail to distinguish nodes that are located in different parts of the graph, but have similar neighborhoods. In the extreme case, if two nodes are located in topologically isomorphic neighborhoods, their learned embeddings are identical. To elaborate, consider nodes v_1 and v_8 in Fig. 1a. The two nodes belong to two different class labels (node color). However, since their 2-hop neighborhoods are isomorphic to each other, their learned embeddings in a 2-layer Graph Convolutional Network (GCN) are identical, despite the fact that they are far away in the graph. Hence, the GCN will be incapable of correctly predicting that v_1 and v_8 belong to different class labels. Consequently, for predictive tasks that rely on position of a node with respect to the graph, the performance suffers. While node embedding methods such as DeepWalk [Perozzi *et al.*, 2014] and Node2Vec [Grover and Leskovec, 2016] do learn position information, they are *transductive* in nature, i.e., they fail to generalize to unseen nodes. In addition, they cannot encode attribute information. Our goal, therefore, is to design a GNN that is inductive, captures both position as well as node attribute information in the embeddings, and overcomes the problem of automorphism leading to identical embeddings.

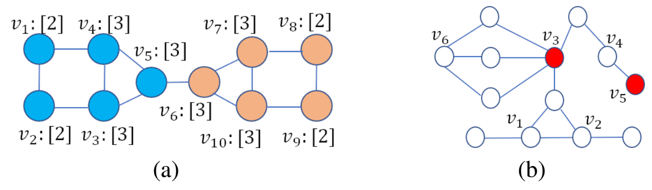


Figure 1: (a) The color of the node indicates its class label. Each node is also characterized by a numerical attribute. (b) The red nodes, v_3 and v_5, represent the anchor nodes.

We emphasize that we do not claim position information to be more important than the neighborhood structure. Rather, the more appropriate choice depends on the problem being solved and the dataset. For example, if homophily is the main driving factor behind a prediction task, then embeddings based on local neighborhood structure are likely to perform well. On the other hand, to predict whether two nodes belong to the same community, e.g., road networks, gene interaction networks, etc., positional information is more important.

P-GNN [You *et al.*, 2019] is the first work to address the need for an inductive GNN that encodes position information. P-GNN randomly selects a small number of nodes as *anchor* nodes. It then computes the shortest path distances of all remaining nodes to these anchor nodes, and embeds this information in a low-dimensional space. Thus, two nodes have similar embeddings if their shortest path distances to the an-

Symbol	Dimensions	Description
$\boldsymbol{\mathcal{X}} = \{\mathbf{x}_v \mid \forall v \in \mathcal{V}\}$	$n \times d$	Original node attributes
$\mathbf{H}^l = \{\mathbf{h}_v^l \mid \forall v \in \mathcal{V}\}$	$n \times d_{hid}$	Feature matrix at layer l
$\boldsymbol{\mathcal{M}}^l = \{\boldsymbol{\mathcal{M}}_v^l \mid \forall v \in \mathcal{V}\}$	$n \times k \times d_{hid}$	Message Tensor at layer l
$\mathbf{W}_{\boldsymbol{\mathcal{M}}}^l$	$2 \cdot d_{hid} \times d_{hid}$	Transform $\boldsymbol{\mathcal{M}}^l$
$\mathbf{a}_{att}^l$	$2 \cdot d_{hid} \times 1$	Attention vector at layer l
$\mathbf{Z} = \{\mathbf{z}_v \mid \forall v \in \mathcal{V}\}$	$n \times k$	Output embeddings
$\mathbf{W}_Z$	$d_{hid} \times 1$	Transforms $\boldsymbol{\mathcal{M}}^L$ to $\mathbf{Z}$

Table 1: Matrix notations and descriptions.

chors are similar. To illustrate, we revisit Fig. 1a, and assume that v_7 is an anchor node. Although the 2-hop neighborhoods of v_1 and v_8 are isomorphic, their shortest path distances to v_7 are different and, hence, their embeddings will be different as well. Although P-GNN has shown impressive improvements over existing GNN architectures in link prediction and pairwise node classification, there is scope to improve.

1. Holistic Position Inference: Since P-GNN relies *only* on shortest path distances to compute embeddings, remaining paths of the graph are ignored. Consequently, two nodes can be incorrectly concluded to be in the same position, even if they are not. Fig. 1b illustrates this aspect. Here, nodes v_1, v_2, and v_6 are all equidistant from the anchors v_3 and v_5 and consequently, their embeddings would be similar. Note, however, that v_6 is located in a different region with several paths of length 2 to v_3, whereas v_1 and v_2 have only one path. Thus, v_6 should have a different embedding.

2. Semantics: The assumption of shortest path rarely holds in the real world. This has been shown across varied domains such as Wordmorph [Gulyás *et al.*, 2018], Wikispeedia [West and Leskovec, 2012], metro transportation [London, 2017], flight connections [TransStat, 2016], etc. Instead, *many* short paths are followed [West and Leskovec, 2012].

3. Robustness to Adversarial Attacks: Relying only on shortest paths also makes P-GNN vulnerable to adversarial attacks. Specifically, adding a small number of critical edges in the graph can significantly alter the shortest path distances for targeted nodes and, hence, their node embeddings.

To overcome these shortcomings, in this paper, we introduce a new position-aware GNN architecture called **GRAPHREACH**[1]. Our key contributions are as follows:

- GRAPHREACH computes position-aware inductive node embeddings through *reachability estimations*. Reachability estimations encode the position of a node with respect to *all* paths in the graph (§2 and §3). This *holistic* position estimation also ensures that GRAPHREACH is robust to adversarial attacks since a few edge additions or deletions do not substantially alter the reachability likelihoods (§4.5).
- Unlike P-GNN, where anchors are randomly selected, GRAPHREACH strategically selects anchors by framing it as a problem of maximizing *reachability* along with their *attention* weights. We show that maximizing reachability is NP-hard, monotone and submodular. We overcome this bottleneck through a greedy hill-climbing algorithm, which provides a $(1-1/e)$ approximation guarantee (§3.2).
- Extensive empirical evaluation across 8 datasets demonstrates a dramatic relative improvement of up to 40% over P-GNN and other state-of-the-art GNN architectures (§4).

[1]Appendices are in https://arxiv.org/abs/2008.09657/.

2 Problem Formulation

We represent a graph as $G = (\mathcal{V}, \mathcal{E}, \boldsymbol{\mathcal{X}}, \mathcal{W})$, where $\mathcal{V}$ is the set of nodes v_i, with $1 \leq i \leq n$ and $\mathcal{E}$ is the set of edges. The attribute set $\boldsymbol{\mathcal{X}} = \{\mathbf{x}_{v_1}, \cdots, \mathbf{x}_{v_n}\}$ has a one-to-one correspondence with the node set $\mathcal{V}$ where $\mathbf{x}_{v_i}$ represents the feature vector of node $v_i \in \mathcal{V}$. Similarly, $\mathcal{W}$ has a one-to-one correspondence with $\mathcal{E}$, where w_{e_i} denotes the edge weights.

A *node embedding model* is a function $f : \mathcal{V} \to \mathbf{Z}$ that maps the node set $\mathcal{V}$ to a d-dimensional vector space $\mathbf{Z} = \{\mathbf{z}_1, \cdots, \mathbf{z}_n\}$. Our goal is to learn a *position-aware* node embedding model [You *et al.*, 2019].

Definition 1 (Position-aware Embedding)**.** *A node embedding $\mathbf{z}_v, \forall v \in \mathcal{V}$ is position-aware if there exists a function $g(\cdot,\cdot)$ such that $d(v_i, v_j) = g(\mathbf{z}_i, \mathbf{z}_j)$, where $d(v_i, v_j)$ is the distance from v_i to v_j in G.*

The distance $d(v_i, v_j)$ should reflect the quality of all paths between v_i and v_j wherein, (1) $d(v_i, v_j)$ is directly proportional to the number of paths between v_i and v_j, and (2) $d(v_i, v_j)$ is inversely proportional to the lengths of the paths between v_i and v_j. We capture these aspects in the form of *reachability estimations* through *random walks*. Note that, $d(\cdot,\cdot)$ is not required to be a metric distance function.

Reachability estimations are similar to PageRank [Brin and Page, 1998] and Random Walk with Restarts [Pan *et al.*, 2004]. To the best of our knowledge, GIL [Xu *et al.*, 2020] is the only other GNN framework that uses the concept of reachability estimations. However, GIL does not use reachability to learn node embeddings. Rather, they are used as additional features along with node embeddings for node classification.

Reachability Estimations: In a fixed-length random walk of length l_w, we start from an initial node v_i, and jump to a neighboring node u through an outgoing edge $e = (v_i, u)$ with *transition probability* $p(e) = w_e / \sum_{\forall e' \in N(v_i)} w_{e'}$. Here, $N(v_i)$ denotes the set of outgoing edges from v_i. From node u, the process continues iteratively in the same manner till l_w jumps. If there are many short paths from v_i to v_j, there is a high likelihood of reaching v_j by a *random walk* from v_i. We utilize this property to define a similarity measure:

$$s(v_i, v_j) = \frac{\sum_{k=1}^{n_w} count_k(v_i, v_j)}{l_w \times n_w} \quad (1)$$

Here, n_w denotes the number of random walks started from v_i, and $count_k(v_i, v_j)$ denotes the number of times random walker visited v_j in the kth random walk starting from v_i. The similarity function could also weight the nodes according to the order they appear in a random walk (refer to App. A). From $s(v_i, v_j)$, one may define $d(v_i, v_j) = 1 - s(v_i, v_j)$. However, we directly work with similarity $s(\cdot,\cdot)$ since $d(v_i, v_j)$ is not explicitly required in our formulation.

3 GRAPHREACH

The symbols and notations used through out this paper are summarized in Table 1. All the matrix and vector notations are denoted in bold letters by convention.

3.1 The Architecture

Algo. 1 outlines the pseudocode and Fig. 2 pictorially depicts the architecture. In addition to hyper-parameters and message

Algorithm 1 GRAPHREACH

```
Input: Graph $G = (\mathcal{V}, \mathcal{E}, \boldsymbol{\mathcal{X}}, \mathcal{W})$; Anchors $\{a_i\}$; Message computation function $\mathcal{F}$; Message aggregation function $\mathcal{S}$; Number of layers $L$; Non-linear function $\sigma$
Output: Node embedding $\mathbf{z}_v, \forall v \in \mathcal{V}$
1: $\mathbf{h}_v^0 \leftarrow \mathbf{x}_v, \forall v \in \mathcal{V}$
2: for $l = 1, \cdots, L$ do
3:     for $v \in \mathcal{V}$ do
4:         for $i = 1, \cdots, k$ do
5:             $\widehat{\boldsymbol{\mathcal{M}}}_v^l[i] \leftarrow \mathcal{F}(v, a_i, \mathbf{h}_v^{l-1}, \mathbf{h}_{a_i}^{l-1})$ ▷ Msg Computation
6:         $\boldsymbol{\mathcal{M}}_v^l = (\oplus_{a_i \in \mathcal{A}} \widehat{\boldsymbol{\mathcal{M}}}_v^l[i]) \cdot \mathbf{W}_{\boldsymbol{\mathcal{M}}}^l$ ▷ Concatenation: Eq. (5)
7:         $\mathbf{h}_v^l \leftarrow \mathcal{S}\left(\boldsymbol{\mathcal{M}}_v^l\right)$ ▷ Msg Aggr: Eq. (6) and Eq. (8)
8: return $\mathbf{z}_v \in \mathbb{R}^k \leftarrow \sigma(\boldsymbol{\mathcal{M}}_v^L.\mathbf{W}_Z), \forall v \in \mathcal{V}$
```

passing functions, the input to Algo. 1 includes the graph and k anchor nodes. The details of the anchor selection procedure are discussed in §3.2. In the initial layer, the embedding of a node v is simply its attribute vector $\mathbf{x}_v$ (line 1). In each hidden layer, a set of *messages*, $\boldsymbol{\mathcal{M}}^l$, is computed using the message computing function $\mathcal{F}(v, a, \mathbf{h}_v^l, \mathbf{h}_a^l)$ between each node-anchor pair (details in §3.3) (lines 2-6). The component $\boldsymbol{\mathcal{M}}_v^l[i]$ in $\boldsymbol{\mathcal{M}}^l$ represents the message received by node v from the i^{th} anchor node. These messages are then aggregated using an aggregation function $\mathcal{S}$ (line 7), which we will discuss in detail in §3.4. The aggregated messages thus obtained are propagated to the next layer. In the final layer, the set of messages for each node, i.e., $\boldsymbol{\mathcal{M}}_v^l$, is linearly transformed through a trainable weight matrix $\mathbf{W}_Z$ (line 8).

3.2 Anchor Selection

Anchors act as our reference points while encoding node positions. It is therefore imperative to select them carefully. Selecting two anchors that are close to each other in the graph is not meaningful since the distance to these anchors from the rest of the nodes would be similar. Ideally, the anchors should be diverse as well as *reachable* from a large portion of nodes.

Formally, let $\mathcal{R}$ be the set of all random walks performed across all nodes in $\mathcal{V}$. We represent the reachability information in the form of a *bipartite* graph $\mathcal{B} = (\mathcal{V}_1, \mathcal{V}_2, \mathcal{E}_B)$. Here, $\mathcal{V}_1 = \mathcal{V}_2 = \mathcal{V}$. There is an edge $e = (u, v) \in \mathcal{E}_B,\ u \in \mathcal{V}_1,\ v \in \mathcal{V}_2$ if there exists a walk in $\mathcal{R}$ that starts from v and *reaches* u. The *reachability set* of a subset of nodes $\mathcal{A}$ is:

$$\rho(\mathcal{A}) = \{v \mid (u, v) \in \mathcal{E}_B,\ u \in \mathcal{A}, v \in \mathcal{V}_2\} \tag{2}$$

Our objective is to find the set of k anchors $\mathcal{A}^* \subseteq \mathcal{V}_1$, $|\mathcal{A}^*| = k$, that *maximizes* reachability. Specifically,

$$\mathcal{A}^* = \underset{\mathcal{A} \subseteq \mathcal{V}_1, |\mathcal{A}| = k}{\arg\max} \{|\rho(\mathcal{A})|\} \tag{3}$$

Lemma 1. *The maximization problem in Eq. 3, performed on the bipartite graph formed on the reachability set, is NP-hard.*

Lemma 2. *For any given set of nodes $\mathcal{A}$, $f(\mathcal{A}) = |\rho(\mathcal{A})|$ is monotone and submodular.*

The proofs are in App. B and App. C respectively.

For *monotone* and *submodular* optimization functions, the greedy-hill climbing algorithm provides a $(1 - 1/e)$ approximation guarantee [Nemhauser *et al.*, 1978]. We, thus, follow the same strategy and iteratively add the node that provides highest *marginal* reachability (Algo. 2 in App. D).

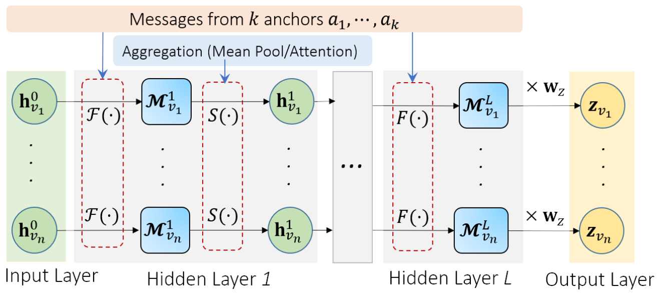


Figure 2: The architecture of GRAPHREACH. Note that aggregation is not performed in the last hidden layer.

Corollary 1. *If set $\mathcal{A}$ is the output of Algo. 2 (in App. D), then $\mid \rho(\mathcal{A}) \mid \geq (1 - 1/e) \mid \mathcal{A}^* \mid$, where $\mathcal{A}^*$ is the anchor set of size k that maximizes reachability coverage.*

PROOF. Follows from Lemma 2.

Modeling Reachability Frequency: The above approach models reachability as a binary occurrence; even if there exists just one walk where $v \in \mathcal{V}_2$ reaches $u \in \mathcal{V}_1$, an edge (u, v) is present in $\mathcal{B}$. It does not incorporate the frequency with which u is visited from v. To capture this aspect, we randomly sample $X\%$ of the walks from $\mathcal{R}$ and form the bipartite graph only on this sampled set. Note that the downsampling does not require the bipartite graph to be fully connected. Algo. 2 (in App. D) is next run to select anchors on this bipartite graph. This process is repeated multiple times by drawing multiple samples of the same size from $\mathcal{R}$ and the final anchor set consists of nodes that are selected in the answer sets the highest number of times. In our experiments, we sample 5 subsets with $X = 30\%$.

3.3 Message Computation

The message computation function $\mathcal{F}\left(v, a, \mathbf{h}_v^l, \mathbf{h}_a^l\right)$ should incorporate both the position information of node v with respect to the anchor set $\mathcal{A}$ as well as the node attributes. While node attributes may provide important information that may be useful in the eventual prediction task, position information, in the form of reachability estimations, captures the location of the node in the global context of the graph. To encode these dual needs, $\mathcal{F}\left(v, a, \mathbf{h}_v^l, \mathbf{h}_a^l\right)$ is defined as follows.

$$\mathcal{F}\left(v, a, \mathbf{h}_v^l, \mathbf{h}_a^l\right) = \left(\left(s(v, a) \times \mathbf{h}_v^l\right) \parallel \left(s(a, v) \times \mathbf{h}_a^l\right)\right) \tag{4}$$

where $\parallel$ denotes concatenation of vectors. The function $\mathcal{F}$ takes as input a node v, an anchor a and their respective layer attributes, $\mathbf{h}_v^l$ and $\mathbf{h}_a^l$. It returns a message vector, which is a weighted aggregation of their individual attributes in proportion to their reachability estimations (Eq. 1). Observe that, the reachability estimations are used in both directions to account for an asymmetric distance function.

Due to concatenation, the output message vector is of dimension $2 \cdot d_{hid}$, where d_{hid} is dimension of the node embeddings in the hidden layers. To linearly transform the message vector back into $\mathbb{R}^{d_{hid}}$, we multiply it with a weight vector $\mathbf{W}_{\boldsymbol{\mathcal{M}}}^l$. The complete global structure information for node v is encompassed in the message matrix $\boldsymbol{\mathcal{M}}_v^l$ ($\oplus$ denotes rowwise stacking of message vectors).

$$\boldsymbol{\mathcal{M}}_v^l = \Big(\bigoplus_{a \in \mathcal{A}} \mathcal{F}\left(v, a, \mathbf{h}_v^l, \mathbf{h}_a^l\right) \Big) \cdot \mathbf{W}_{\boldsymbol{\mathcal{M}}}^l \tag{5}$$

3.4 Message Aggregation

To compute the hidden representation of nodes, messages corresponding to anchors are aggregated for each node. We propose two aggregation schemes.

1. Mean Pool (M): In this, a simple mean of the message vectors are taken across anchors.

$$\mathcal{S}^M(\boldsymbol{\mathcal{M}}_v^l) = \frac{1}{k} \sum_{i=1}^{k} \boldsymbol{\mathcal{M}}_v^l[i] \tag{6}$$

2. Attention Aggregator (A): In mean pooling, all anchors are given equal weight. Theorizing that the information being preserved can be enhanced by capturing the significance of an anchor with respect to a node, we propose to calculate the significance distribution among anchors for each node. Following the Graph Attention Network (GAT) architecture [Velickovic *et al.*, 2018], we compute attention coefficients of anchors for an anchor-based aggregation. The attention coefficient of the i^{th} anchor a_i is computed with trainable weight vector $\mathbf{a}_{att}^l$ and weight matrix $\mathbf{W}_{att}^l$. For node v, the attention weight with respect to anchor i is

$$\alpha_v[i] = \mathrm{sm}\Big(\sigma_{att}\Big((\mathbf{h}_v^l \cdot \mathbf{W}_{att}^l \parallel \boldsymbol{\mathcal{M}}_v^l[i] \cdot \mathbf{W}_{att}^l) \cdot \mathbf{a}_{att}^l\Big)\Big) \tag{7}$$

Here, sm denotes the softmax function. As followed in GAT architecture [Velickovic *et al.*, 2018], we use *LeakyReLU* as the non-linear function σ_{att} (with negative input slope 0.2).

Finally, the messages are aggregated across anchors using these coefficients.

$$\mathcal{S}^A(\boldsymbol{\mathcal{M}}_v^l) = \sum_{i=1}^{k} \alpha_v[i] \times \boldsymbol{\mathcal{M}}_v^l[i] \cdot \mathbf{W}_{att}^l + \mathbf{h}_v^l \cdot \mathbf{W}_{att}^l \tag{8}$$

3.5 Hyper-parameters for Reachability Estimation

Reachability information relies on two key random walk parameters: the length l_w of each walk, and the total number of walks n_w. If l_w is too short, then we do not gather information with respect to anchors more than l_w-hops away. With n_w, we allow the walker to sample enough number of paths so that our reachability estimations are accurate and holistic. We borrow an important theorem from [Wadhwa *et al.*, 2019] to guide our choice of l_w and n_w.

Theorem 1. *[Wadhwa* et al.*, 2019] If there exists a path between two nodes u and v in a graph, with $1 - 1/n$ probability the random walker will find the path if the number of random walks conducted, n_w, is set to $\Theta(\sqrt[3]{n^2 \ln n})$ with the length of each random walk, l_w, being set to the diameter of the graph.*

3.6 Complexity Analysis

We conduct n_w random walks of length l_w for all the n nodes of the graph; this requires $O(n_w \cdot l_w \cdot n)$ time. For anchor set selection, we sample random walks multiple times and select k anchors using the resulting bipartite graphs formed. The complexity of sampling walks is $O(n \cdot n_w)$ while selecting the anchors takes $O(k + k \cdot \log k) = O(k \cdot \log k)$ operations. Considering that each node communicates with k anchors, there are $O(n \cdot k)$ message computations. The aggregation of messages also requires $O(n \cdot k)$ operations owing to k messages being aggregated for each of the n nodes. The attention aggregator has an additional step devoted to computation of attention coefficients which takes $O(n \cdot k)$ time as well.

Datasets	**#Nodes**	**#Edges**	**#Labels**	**Diameter**	**#Attributes**
Grid	400	760	-	38	-
Communities	400	3.8K	20	9	-
PPI	56.6K	818K	-	8	50
Email-Complete	986	16.6K	42	7	-
Email	920	7.8K	6	7	-
Protein	43.4K	81K	3	64	29
CoRA	2.7K	5.4K	7	19	1433
CiteSeer	3.3K	4.7K	6	28	3703

Table 2: Characteristics of graph datasets used.

4 Experiments

In this section, we benchmark GRAPHREACH and establish that GRAPHREACH provides up to 40% relative improvement over state-of-the-art GNN architectures. The implementation is available at https://github.com/idea-iitd/GraphReach.

4.1 Experimental Setup

Please refer to App. E for information on reproducibility.

Datasets: We evaluate GRAPHREACH on the datasets listed in Table 2. Further details are available in App. E.

Baselines: We measure and compare the performance of GRAPHREACH with five baselines: (1) P-GNN [You *et al.*, 2019], (2) GCN [Kipf and Welling, 2017], (3) GRAPHSAGE [Hamilton *et al.*, 2017], (4) GAT [Velickovic *et al.*, 2018], and (5) GIN [Xu *et al.*, 2019].

Prediction Tasks: We evaluate GRAPHREACH on the prediction tasks of *Link Prediction (LP)* and *Node Classification (NC)* using the *Binary Cross Entropy* (BCE) loss with *logistic activation*, and on *Pairwise node classification (PNC)* using the *Negative Log Likelihood* (NLL) loss.

Setting: For LP, we evaluate in both inductive and transductive settings, whereas for PNC , only inductive setting is used.

Default Parameters and Design Choices: Unless specifically mentioned, we set the number of anchors (k) as $\log^2 n$. While our experiments reveal that a smaller number of anchors is sufficient, since P-GNN uses $\log^2 n$ anchors, we keep it the same. The length of each random walk, l_w, is set to $graph\ diameter$, and the number of walks n_w as 50. We also conducted experiments to analyze how these parameters influence prediction accuracy of GRAPHREACH.

We use *Attention* aggregation (Eq. 8) and simple random walk counts as the similarity function (Eq. 1) to compare with the baselines. For a fair comparison, values of parameters that are common to GRAPHREACH and other baselines are kept the same. Their exact values are reported in App. E.

4.2 Comparison with Baselines

Pairwise Node Classification (PNC): Table 3a summarizes the performances in PNC. We observe a dramatic performance improvement by GRAPHREACH over all existing GNN architectures. While P-GNN clearly established that encoding global positions of nodes helps in PNC, GRAPHREACH further highlights the need to go beyond shortest paths. Except

Models	Communities	Email	Email-Complete	Protein
GCN	0.520 ± 0.025	0.515 ± 0.019	0.536 ± 0.006	0.515 ± 0.002
GRAPHSAGE	0.514 ± 0.028	0.511 ± 0.016	0.508 ± 0.004	0.520 ± 0.003
GAT	0.620 ± 0.022	0.502 ± 0.015	0.511 ± 0.008	0.528 ± 0.011
GIN	0.620 ± 0.102	0.545 ± 0.012	0.544 ± 0.010	0.523 ± 0.002
P-GNN-F	0.997 ± 0.006	0.640 ± 0.037	0.630 ± 0.031	0.729 ± 0.176
P-GNN-E	$\mathbf{1.000 \pm 0.001}$	0.640 ± 0.029	0.637 ± 0.037	0.631 ± 0.175
GRAPHREACH	$\mathbf{1.000 \pm 0.000}$	$\mathbf{0.949 \pm 0.009}$	$\mathbf{0.935 \pm 0.006}$	$\mathbf{0.904 \pm 0.003}$

(a) Pairwise Node Classification (PNC)

Models	Grid-T	Communities-T	Grid	Communities	PPI
GCN	0.698 ± 0.051	0.981 ± 0.004	0.456 ± 0.037	0.512 ± 0.008	0.769 ± 0.002
GRAPHSAGE	0.682 ± 0.050	0.978 ± 0.003	0.532 ± 0.050	0.516 ± 0.010	0.803 ± 0.005
GAT	0.704 ± 0.050	0.980 ± 0.005	0.566 ± 0.052	0.618 ± 0.025	0.783 ± 0.004
GIN	0.732 ± 0.050	0.984 ± 0.005	0.499 ± 0.054	0.692 ± 0.049	0.782 ± 0.010
P-GNN-F	0.637 ± 0.078	0.989 ± 0.003	0.694 ± 0.066	$\mathbf{0.991 \pm 0.003}$	0.805 ± 0.003
P-GNN-E	0.834 ± 0.099	0.988 ± 0.003	0.940 ± 0.027	0.985 ± 0.008	0.808 ± 0.003
GRAPHREACH	$\mathbf{0.945 \pm 0.021}$	$\mathbf{0.990 \pm 0.005}$	$\mathbf{0.956 \pm 0.014}$	$\mathbf{0.991 \pm 0.003}$	$\mathbf{0.810 \pm 0.002}$

(b) Link Prediction (LP)

Table 3: ROC AUC. (Grid-T and Communities-T indicate performance in transductive settings. P-GNN-E uses exact shortest path distance to all anchors while P-GNN-F is a fast variant of P-GNN that uses truncated 2-hop shortest path distance.)

Task	Dataset	GCN		GRAPHSAGE		GAT		GIN		P-GNN		GRAPHREACH	
		S+T	S	S+T	S	S+T	S	S+T	S	S+T	S	S+T	S
LP	CoRA	**0.86**	0.59	0.85	0.53	**0.86**	0.51	**0.86**	0.59	0.81	0.77	0.83	**0.84**
LP	CiteSeer	**0.87**	0.61	0.85	0.56	**0.87**	0.56	0.86	0.68	0.77	**0.76**	0.77	0.74
PNC	CoRA	**0.98**	0.50	0.96	0.50	**0.98**	0.51	**0.98**	0.52	0.86	0.59	0.96	**0.77**
PNC	CiteSeer	**0.96**	0.51	0.95	0.51	**0.96**	0.50	**0.96**	0.53	0.77	0.57	0.91	**0.61**
NC	CoRA	**0.92**	0.52	**0.92**	0.50	0.90	0.50	0.90	0.54	0.73	0.50	0.84	**0.86**
NC	CiteSeer	**0.82**	0.52	**0.82**	0.50	0.81	0.50	**0.82**	0.53	0.73	0.55	0.75	**0.71**

Table 4: ROC AUC. ('S' denotes the version containing only the graph structure and 'S+T' denotes structure with node attributes.)

in Communities, the highest accuracy achieved by any of the baselines is 0.73. In sharp contrast, GRAPHREACH pushes the ROC AUC above 0.90, which is a significant $\approx 40\%$ relative improvement, on average, over the state-of-the-art.

Link Prediction (LP): Table 3b presents the results for LP. Consistent with the trend observed in PNC, GRAPHREACH outperforms other GNNs across inductive and transductive settings. P-GNN and GRAPHREACH are significantly better than the rest of the architectures. This clearly indicates that position-aware node embeddings help. GRAPHREACH being better than P-GNN suggests that a holistic approach of encoding position with respect to all paths is necessary.

The second observation from Table 3b is that the performance of position-*unaware* architectures are noticeably better in transductive setting. Since transductive setting allows unique identification of nodes through one-hot encodings, traditional GNN architectures are able to extract some amount of position information, which helps in the prediction. In contrast, for both P-GNN and GRAPHREACH, one-hot encodings do not impart any noticeable advantage as position information is captured through distances to anchors.

We also compare the time taken by the two best performing models (in App. F). On average, GRAPHREACH is 2.5 times faster than P-GNN owing to its strategic anchor selection.

4.3 Difference from Neighborhood-based GNNs

We conducted experiments on attributed graph datasets, with and without attributes for prediction tasks. Table 4 presents the results for CoRA and CiteSeer on LP, PNC and NC.

In addition to the network structures, both CoRA and CiteSeer, are also accompanied by binary word vectors characterizing each node. When the word vectors are ignored, the performance of neighborhood-aggregation based GNNs are significantly inferior ($\approx 25\%$) to GRAPHREACH. When supplemented with word vectors, they outperform GRAPHREACH ($\approx 10\%$ better). This leads to the following conclusions. (1) Position-aware GNNs are better in utilizing the structural information. (2) Neighborhood-aggregation based GNNs may be better in exploiting the feature distribution in the neighborhood. (This is not always though, e.g., in PPI and Protein, GRAPHREACH is better than the baselines even when attribute information is used as seen in Tables 3a and 3b). (3) The two approaches are *complementary* in nature and, therefore, good candidates for ensemble learning. To elaborate, neighborhood aggregation based architectures rely on *homophily* [Zhu *et al.*, 2020]. Consequently, if the node property is a function of neighborhood attribute distribution, then neighborhood aggregation performs well. On the other hand, in LP, even if the local neighborhoods of two distant nodes are isomorphic, this may not enhance their chance of having a link. Rather, the likelihood increases if two nodes have many neighbors in common. When two nodes have common neighbors, their distances to the anchors are also similar, and this positional information leads to improved performance.

4.4 Ablation Study: Mean Pool versus Attention

In the first two rows of Table 5, we compare the performance of GRAPHREACH with Attention (GRAPHREACH-A) and Mean Pool (GRAPHREACH-M) aggregation functions. As clearly evident, the performance is comparable. In mean pool, the message from each anchor is weighted equally, while when an attention layer is used, GRAPHREACH learns an additional importance weight for each anchor to aggregate messages. The comparable performance of Mean Pool with Attention shows that similarity encoded in the message from each anchor is enough to learn meaningful embeddings; the marginal gain from an additional attention layer is minimal. To further substantiate this claim, we alter Eq. 4 to $\mathcal{F}\left(v, a, \mathbf{h}_v^l, \mathbf{h}_a^l\right) = \mathbf{h}_v^l \parallel \mathbf{h}_a^l$; i.e, contributions of all anchors are made equal. The variant GRAPHREACH-M$^-$ presents the performance with this modification; as evident, there is a massive drop in accuracy. We also evaluate GRAPHREACH using a similarity function variant (in App. F).

4.5 Adversarial Attacks

We assume the standard *black-box* adversarial setup where the attacker has knowledge of only the graph and can modify it through addition or deletion of edges [Li *et al.*, 2020; Chang *et al.*, 2020]. Let $G = (\mathcal{V}, \mathcal{E})$ be the test graph which has a *colluding* subset of nodes $\mathcal{C} \subseteq \mathcal{V}$. The nodes in $\mathcal{C}$ can add as many edges as needed among them so that predictions on $\mathcal{C}$ are inaccurate. For PNC, we randomly sample 10% nodes from the unseen test graph, and form a clique among these colluding nodes. For LP, we randomly sample 10% of the node pairs from the unseen test graph such that they do

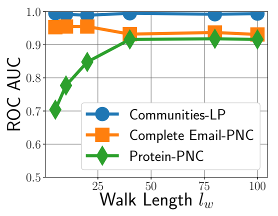
Models	Communities	Email	Email-Complete	Protein
GR-A	$\mathbf{1.000 \pm 0.000}$	$\mathbf{0.949 \pm 0.009}$	0.935 ± 0.006	0.904 ± 0.003
GR-M	$\mathbf{1.000 \pm 0.000}$	0.938 ± 0.017	$\mathbf{0.945 \pm 0.004}$	$\mathbf{0.916 \pm 0.008}$
GR-M$^-$	0.500 ± 0.000	0.500 ± 0.000	0.500 ± 0.000	0.559 ± 0.007

(a) Pairwise Node Classification (PNC)

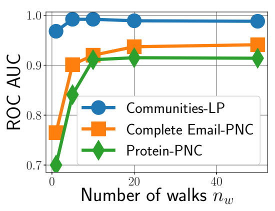
Models	Grid-T	Communities-T	Grid	Communities	PPI
GR-A	$\mathbf{0.945 \pm 0.021}$	0.990 ± 0.005	$\mathbf{0.956 \pm 0.014}$	0.991 ± 0.003	0.810 ± 0.002
GR-M	0.940 ± 0.018	$\mathbf{0.994 \pm 0.003}$	0.931 ± 0.020	$\mathbf{0.993 \pm 0.003}$	$\mathbf{0.830 \pm 0.004}$
GR-M$^-$	0.542 ± 0.071	0.888 ± 0.046	0.500 ± 0.000	0.500 ± 0.000	0.519 ± 0.026

(b) Link Prediction (LP)

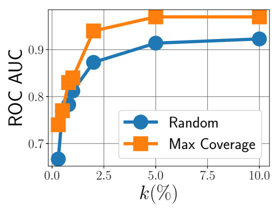
Table 5: ROC AUC in PNC and LP for ablation study. (GR stands for GRAPHREACH.)



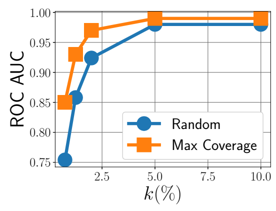
(a) Random Walk Length, l_w



(b) Number of Walks, n_w



(c) PNC (Email-Complete)



(d) LP (Communities)

Figure 3: Impact of (a) walk length (b) number of walks and (c-d) number of anchors for PNC and LP, on accuracy of GRAPHREACH.

Dataset	Task	P-GNN			GRAPHREACH		
		Bf	Af	Δ	Bf	Af	Δ
Communities	PNC	0.92	0.82	0.10	1.00	0.98	**0.02**
Communities	LP	1.00	0.89	0.11	1.00	1.00	**0.00**

Table 6: Robustness to adversarial attacks. (*Bf* and *Af* indicate ROC AUC before and after collusion respectively, while Δ denotes the change in accuracy due to collusion.)

not have an edge between them. From this set, we select the top-2% of the highest degree nodes and connnect them to the remaining colluding nodes through an edge. This makes the diameter of the colluding group at most 2.

We perform prediction using pre-trained models of both P-GNN and GRAPHREACH on the test graph and measure the ROC AUC on the colluding group. This process is repeated with 5 different samples of colluding groups and the mean accuracy is reported. If the model is robust, despite the collusion, its prediction accuracy should not suffer.

As visible in Table 6, the impact on the accuracy of GRAPHREACH is minimal. On the other hand, P-GNN receives more than 10% relative drop in the performance. This result highlights one more advantage of reachability estimations. Since GRAPHREACH incorporates all paths in its position-aware embeddings, causing a significant perturbation in position with respect to all paths in the network is difficult. In case of P-GNN, perturbing shortest paths is relatively easier and hence there is a higher impact on the performance.

4.6 Impact of Parameters

Random Walk Length, l_w: Fig. 3a presents the result on three datasets covering both PNC and LP. Results on remaining datasets are provided in App. F. On Communities and Email-Complete, the impact of l_w is minimal. However, in Protein, the accuracy saturates at around $l_w = 40$. This is a direct consequence of the property that Protein has a significantly larger diameter of 64 than both Communities and Email-Complete (see Table 2). Recall from our discussion in §3.5 that setting l_w to the graph diameter is recommended for ensuring accurate reachability estimations. The trends in Fig. 3a substantiate this theoretical result.

Number of Random Walks, n_w: Fig. 3b presents the results. As expected, with higher number of walks, the accuracy initially improves and then saturates. From §3.5, we know that n_w should increase with the number of nodes in the graph to ensure accurate reachability estimations. The trend in Fig. 3b is consistent with this result. In App. F, we present the results for this experiment on the remaining datasets.

Number of Anchors and Anchor Selection Strategy: Fig. 3(c-d) present the results. In addition to the default anchor selection strategy, we also evaluate the accuracy obtained when an equal number of anchors are selected randomly. Two key properties emerge from this experiment. First, as the number of anchors increases, the accuracy improves till a saturation point. This is expected since with more anchors we have more reference points to accurately encode node positions. Second, the proposed anchor selection strategy is clearly better than random anchor selection. More importantly, the proposed anchor selection strategy saturates at around 2.5% compared to 5% in random. Recall that the dimension of the final embeddings, i.e., the final layer, is equal to the number of anchors. Consequently, this experiment highlights that high quality embeddings can be obtained within a low-dimensional space. A low dimension is preferred in various tasks such as indexing and querying of multi-dimensional points, due to the adverse impacts of *curse-of-dimensionality* [Samet, 2006].

5 Conclusions

GNN architectures, despite their impressive success in learning node embeddings, suffer on predictive tasks that rely on positional information. Though P-GNN recognized this need, due to reliance on only shortest paths, the position information captured by P-GNN is not holistic. GRAPHREACH addresses this limitation and builds a different positional model based on reachability estimations to anchors computed strategically through fixed-length random walks. GRAPHREACH achieves a relative improvement of up to 40% over other architectures, while also being robust to adversarial attacks.

References

[Brin and Page, 1998] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks*, 30:107–117, 1998.

[Chang *et al.*, 2020] Heng Chang, Yu Rong, Tingyang Xu, Wenbing Huang, Honglei Zhang, Peng Cui, Wenwu Zhu, and Junzhou Huang. A restricted black-box adversarial framework towards attacking graph embedding models. In *AAAI*, pages 3389–3396, 2020.

[Goyal *et al.*, 2020] Nikhil Goyal, Harsh Vardhan Jain, and Sayan Ranu. Graphgen: A scalable approach to domain-agnostic labeled graph generation. In *Proceedings of The Web Conference 2020*, pages 1253–1263, 2020.

[Grover and Leskovec, 2016] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *ACM SIGKDD*, pages 855–864, 2016.

[Gulyás *et al.*, 2018] András Gulyás, Zalán Heszberger, József Bíró, János Tapolcai, Attila Csoma, István Pelle, Attila Kőrösi, Dávid Klajbár, Valentina Halasi, Gábor Rétvári, and Márton Novák. A dataset on human navigation strategies in foreign networked systems. *Scientific Data*, 5, 03 2018.

[Hamilton *et al.*, 2017] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS 2017*, pages 1024–1034, 2017.

[Kipf and Welling, 2017] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th ICLR*, 2017.

[Li *et al.*, 2020] Jia Li, Honglei Zhang, Zhichao Han, Yu Rong, Hong Cheng, and Junzhou Huang. Adversarial attack on community detection by hiding individuals. In *WWW 2020*, page 917–927, 2020.

[London, 2017] Transport London. Rolling Origin and Destination Survey, 2017.

[Manchanda *et al.*, 2020] Sahil Manchanda, Akash Mittal, Anuj Dhawan, Sourav Medya, Sayan Ranu, and Ambuj Singh. GCOMB: Learning budget-constrained combinatorial algorithms over billion-sized graphs. *Advances in Neural Information Processing Systems*, 33, 2020.

[Nemhauser *et al.*, 1978] George Lann Nemhauser, Laurence Alexander Wolsey, and Marshall Latham Fisher. An analysis of approximations for maximizing submodular set functions-I. *Mathematical Programming*, 14(1):265–294, 1978.

[Pan *et al.*, 2004] Jia-Yu Pan, Hyung-Jeong Yang, Christos Faloutsos, and Pinar Duygulu. Automatic multimedia cross-modal correlation discovery. In *ACM SIGKDD*, pages 653–658, 2004.

[Perozzi *et al.*, 2014] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. DeepWalk: Online learning of social representations. In *ACM SIGKDD*, pages 701–710, 2014.

[Samet, 2006] Hanan Samet. *Foundations of multidimensional and metric data structures*. Academic Press, 2006.

[TransStat, 2016] RITA TransStat. Origin and Destination Survey database (DB1B), 2016.

[Velickovic *et al.*, 2018] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *6th ICLR*, 2018.

[Wadhwa *et al.*, 2019] Sarisht Wadhwa, Anagh Prasad, Sayan Ranu, Amitabha Bagchi, and Srikanta Bedathur. Efficiently answering regular simple path queries on large labeled networks. In *ICMD*, pages 1463–1480, 2019.

[West and Leskovec, 2012] Robert West and Jure Leskovec. Human wayfinding in information networks. In *WWW*, page 619–628, 2012.

[Xu *et al.*, 2019] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *7th ICLR*, 2019.

[Xu *et al.*, 2020] Chunyan Xu, Zhen Cui, Xiaobin Hong, Tong Zhang, Jian Yang, and Wei Liu. Graph inference learning for semi-supervised classification. In *8th ICLR*, 2020.

[You *et al.*, 2019] Jiaxuan You, Rex Ying, and Jure Leskovec. Position-aware graph neural networks. In *ICML*, pages 7134–7143, 2019.

[Zhu *et al.*, 2020] Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. In *NeurIPS*, 2020.