An Augmented Neighborhood Network for Graph Representation Learning

Boning Zheng, Carson Zhao
Department of Statistics
Stanford University
{b7zheng,czhao333}@stanford.edu

Abstract

In this paper, we explore an extended neighborhood approach to two graph representation learning methods, GraphSAGE and node2vec. We propose a novel model that artificially augments a node's neighborhood to capture information from outside a node's local neighborhood. Our augmented neighborhood network (ANN) exploits structural information to improve the performance of GraphSAGE and node2vec. Our model improves node classification performance on three synthetic datasets generated from various power law distributions, and also upon a real-life molecular structure dataset. We also propose a measure of structural equivalence, which we call the structural coefficient. From our experiments, we found that the higher the structural coefficient of a network, the more indicative structural roles are for node classification.

1. Introduction

Using vector embeddings of nodes for network analysis have proven to be very successful for a variety of tasks such as node classification and link prediction. Learning representations for nodes in networks can be done with models such as node2vec and GraphSAGE. In this paper, we aim to adapt these node embedding methods to include richer structural information. First, we propose a new measure for structural equivalence in the context of node classification. Then based on these measures, we plan to adapt node2vec and GraphSAGE to better take advantage of this structural information.

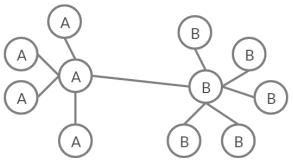

Figure 1. Node features influenced by homophily

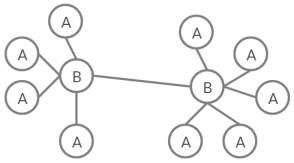

Figure 2. Node features influenced by structural equivalence

In real-life networks, we see nodes being influenced by a fair mix of both homophily and structural equivalence. For example in Figure 1, the type of the node (A or B) is dictated mostly by homophily. On the other hand in Figure 2, the node type is dictated mostly by its structural role in the graph. For the purposes of node classification, we would want the feature vectors of all the nodes of one class to be similar. So in this example, all nodes of class type A should have similar features vectors, which will be different than the feature vectors of class type B nodes.

A similar weakness in both node2vec and GraphSAGE is that both do not exploit structural equivalence. To address this, we propose a novel way of augmenting the neighborhood search algorithm to jump to other nodes that are structurally equivalent across the graph. If structural equality does play a strong role in determining the features of a node, then the feature vector of two structurally equivalent nodes should be similar as well. Therefore, re-defining the neighborhood of a target node to be its own neighbors plus nodes that are structurally equivalent could enrich the feature information of the target node.

One caveat of GraphSAGE is that it only considers features of its local neighbours. This is good if the graph exhibits high homophily. However if the node's features are more dependent on the structural properties of the node, then it makes more sense to define the "neighborhood" of the node to be nodes that are structurally similar to it, rather than nodes that are close to it. Most networks exhibit a fair balance of homophily and structural equivalence. We believe that incorporating feature information from nodes that are structurally similar as well can improve the power and performance of the algorithms.

2. Related Work

In this section, we will first discuss the RolX role discovery algorithm. RolX allows us to find structurally similar nodes to any query node. We utilize this to construct the augmented neighborhood. Then, we discuss two representation learning models, node2vec and GraphSage.

In Henderson et al 2012 [3], the authors explored the concept of automatically identifying structural roles of nodes in a network based on structural features of itself and its neighborhood. The features are aggregated recursively and include egonet information. Through their experiments, the roles identified are effective in exploratory data analysis, similar node search, and in network prediction tasks such as node classification. This algorithm enables us to classify each node to a role that it most likely represents, and then gives us a feature vector for that node that is representative of its role. RolX consists of feature extraction, feature grouping, and model section. The algorithm is described in more detail in Section 3.1.

In Grover and Leskovec 2016 [1], the authors introduced the node2vec algorithm. Its goal is to learn a feature vector for each node that maximizes the likelihood of preserving network neighborhoods of nodes. For example if nodes u and v are a part of the same neighborhood, then the feature vector for nodes u and v should be similar. A node's neighborhood is explored using a biased random walk that can capture both a local and macro-level view of the graph. The algorithm is discussed further in Section 3.2.

GraphSAGE was introduced in Hamilton et al 2017 [2], and is one of the current state-of-the-art algorithms in learning low-dimensional embeddings of nodes. It is an expansion on graph convolutional networks (GCNs) [4] by Kipf et al. GCNs generate node embeddings based on local network neighborhoods by encoding graph structure and aggregating node information with neural networks. GraphSAGE adds a generalized neighbor aggregation mechanism. The algorithm holds top results in node classification across multiple categories of network types. The success of GraphSAGE comes from how it leverages the node's feature information as well as the feature information of its neighborhood nodes to come up with a feature vector. It also has an inductive capability, which means it can generalize to new nodes and graphs.

3. Methods

3.1. RolX

RolX can be summarized in the following steps.

- *Feature Extraction*
  In this step, RolX describes each node as a feature vector. These features can include local information such as the number of neighbors the node has, how many edges in a node's egonet, number of triangles the node takes part in, etc. It also computes recursive features aggregated from its direct neighbors and neighbors of neighbors, recursively expanding to larger neighborhoods. These features can be aggregated by using an aggregate function such as the mean or the max.

- *Feature Grouping*
  After feature extraction, we have n vectors of size f (number of features), which we store in $V_{n\times f}$. We then seek two low rank matrices $G_{n\times r}$ and $F_{r\times f}$ such that they satisfy $\text{argmin}_{G,F} \parallel V - GF \parallel_F$ where $\parallel \cdot \parallel_F$ is the Frobenius norm. Each row of $G_{n\times r}$ represents a node's membership for each role and each column of $F_{r\times f}$ specifies how each role contributes to the value of the node's feature vector.

- *Model Selection*
  Here, the best model is chosen, e.g. how many roles and features are selected. The authors choose to select the model that minimizes the sum of the description cost M and the coding cost $\mathcal{E}$. M is the total space in bits to store the matrices G and F, and $\mathcal{E}$ is the KL divergence of V to GF.

3.2. node2vec

To characterize a neighborhood of a node, the node2vec algorithm takes multiple biased random walks from the node and adds all the nodes it visits to its neighborhood. To control the search bias of the random walk, the algorithm contains two tunable parameters: p, the return parameter and q, the in-out parameter. The parameter p controls the likelihood of revisiting nodes and the parameter q controls the likelihood between going inward or going outward relative to the source node. With both of these parameters, we can control whether we explore in a more breadth-first search way or more of a depth-first search way. If $q < 1$, then the random walk is more reflective of DFS behavior, and if $q > 1$, the random walk obtains a local view of the graph and is reflective of a BFS.

After we determine the neighborhoods, we choose feature vectors f for the nodes that maximize the following objective function:

$$\max_f \sum_{u\in V} \log Pr(N_S(u)|f(u))$$

where V is the vertex set and $N_S(u)$ represents the neighborhood of u determined from the search strategy S. The objective is optimized using stochoastic gradient ascent and negative sampling, which is explained in more detail in the paper.

3.3. GraphSAGE

The embedding generation algorithm consists of K aggregation functions, which aggregates the features of K levels of the node's neighbors. At each step, each node aggregates the representations of the node's immediate neighborhood. Then this aggregated neigborhood vector is concatenated to the node's current representation and fed through a fully connected layer with a nonlinearity σ.

$$h_v^k \leftarrow \sigma(\mathbf{W}^k \cdot \text{CONCAT}(h_v^{k-1}, AGGREGATE_k(h_u^{k-1})))$$

where $k \in \{1, ..., K\}$, $u \in N(v)$, and v is a node in the graph. One example of an aggregation function is the mean function, which takes the mean of the features of a node's neighbours. The feature vector for the k^{th} layer would then be represented by

$$h_v^k \leftarrow \sigma(\mathbf{W} \cdot \text{MEAN}(\{h_v^{k-1}\} \cup \{h_u^{k-1}, \forall u \in N(v)\}))$$

The parameters $\mathbf{W}$ for the aggregator functions could be learned via stochastic gradient descent on a specified loss objective, for example logloss for node classification.

3.4. Augmented Neighborhood Network

We now describe a novel way of augmenting the neighborhood search algorithm to jump to other nodes across the graph that are structurally equivalent, which we abbreviate as ANN.

We leverage the RolX feature extractor that we implemented using our selected features. The basic features included in our implementation are the node degree, edges in the egonet of node, the cut of the egonet (number of edges that enter or leave the egonet), and the node clustering coefficient. The feature matrix $X_f \in \mathbb{R}^{|V| \times 4^{L+1}}$ is constructed recursively where L is the number of recursive iterations and $|V|$ is the number of nodes in the graph. The recursive aggregators are sum, mean, and standard deviation. Next, we prune the features matrix by finding pairs of columns that have perfect correlation and for each pair, deleting one of the perfectly correlated columns. Afterwards we calculate the cosine similarity of each node's features to each other node's features resulting in a $S \in \mathbb{R}^{|V| \times |V|}$ similarity matrix.

We find the top K_v similar nodes for each node v in S, and store this in an adjacency list, where each row refers to a node's top K_v most similar nodes by node ID. K_v can be chosen to be the same for all nodes, or a value proportional to the out-degree of a node which we define to be the **augmentation ratio** α.

In terms of specific algorithm adaptions, we adjust GraphSAGE and node2vec in the following ways. For GraphSAGE, we perform the neighborhood augmentations by concatenating a set of directed edges to the original edge list. For every node v, we add in directed edges from v to its most structurally similar nodes from RolX.

For node2vec, we add the most structurally similar nodes as defined above to the neighborhood of each node. Thus, there is a proportional probability that the random walk will visit the augmented neighbors instead of the original neighbors.

The RolX feature extraction is done only once in the data preprocessing step. Augmenting the edge list in GraphSAGE only introduces $K_v \times |V|$ more edges to the edge list, so we don't expect the run-time to be significantly affected. Similarly, the run-time of node2vec should only increase proportionally with the number of edges added.

Algorithm 1: ANN algorithm

Input : Graph $G(V, E)$;
input features $\{\mathbf{x}_v \forall v \in V\}$;
model type $F \in \{$node2vec, GraphSAGE$\}$;
augmentation ratio α
Output: Vector representations $\mathbf{z}_v$ $\forall v \in V$
$X_f \leftarrow$ *RolX*$(G(V, E))$;
$\tilde{E} \leftarrow E$;
for $v \in V$ **do**
  $e \leftarrow \emptyset$;
  for $i = 1, ..., \alpha k_v$ **do**
    $e \leftarrow e +$ *Augment*(i, v, X_f)
  end
  $\tilde{E} \leftarrow \tilde{E} + e$
end
$\mathbf{z}_v \leftarrow F(v, \mathbf{x}_v, G(V, \tilde{E}))$ $\forall v \in V$;

where X_f is the recursive RolX feature matrix, k_v is the node degree of node v, $\mathbf{x}_v$ is the RolX feature vector for node v, $\tilde{E}$ is the augmented edge list, and *Augment* returns the i-th most structurally similar node to v based on X_f.

3.5. Structural Coefficient

We now describe a new measure for structural equivalence, called the **structural coefficient**. Given a graph and labels corresponding to each node, we calculate the structural coefficient ψ as follows. We first calculate the RolX features to get $\mathbf{X} \in \mathbb{R}^{N \times M}$, where $M = 4^{L+1}$ is the length of the RolX feature vector and $N = |V|$ is the total number of nodes. Each row X_i represents the features for node i. Let C denote the number of distinct class labels, and $S_1, ..., S_C$ represent the C groups of nodes with $N_c = |S_c|$. Furthermore let $\overline{X_c} = \frac{1}{N_c} \sum_{i \in S_c} X_i$ be the average feature vector for label c.

We define the structural coefficient of a Graph to be

$$\psi = \frac{Var(\{\overline{X_1}, ..., \overline{X_C}\})}{\sum_{c=1}^{C} w_c Var(\{X_i; i \in S_c\})}$$

where we define the weights to be $w_c = \frac{N_c}{N}$ and the variance of a set of vectors to be

$$Var(\{X_1, ..., X_n\}) = \frac{1}{M} \sum_{j=1}^{M} Var(X_{1j}, X_{2j}, ..., X_{nj})$$

The numerator is a measure for inter-class variation and the denominator is a weighted average of intra-class variations. Therefore the structural coefficient can be interpreted as a ratio of between-class variation to within-class variation. The range of the metric is $[0, +\infty)$. 0 is the lower bound since variance is non-negative. If structural roles are highly indicative towards node classification, then we expect a high value for the structural coefficient. Conversely if the structural coefficient is high for a graph, then the node labels will depend largely on the structure of the nodes.

One limitation of this proposed measure is that node labels are necessary to compute the measure. It cannot be calculated in the unsupervised setting.

4. Data

We use a variety of open-source datasets as well as simulated graphs of varying structures.

- **CiteSeer** and **Cora**
  CiteSeer is an online digital library for scientific papers [5]. The nodes represent publications and the directed edges represent citations. The labels represent the area of research the paper is about. There are 6 different classes, 3,327 nodes, and 4,732 edges. Cora is another citation network dataset where nodes are documents and edges are citation links [5]. There are 7 classes, 2,708 nodes, and 5,429 edges.

- **BZR**
  The BZR dataset [6] consists of a collection of 405 distinct ligands for the benzodiazepine receptor. The nodes represent atomic particles and the edges represents links between them. The labels represent the type of atomic particle. There are 7 classes, 14,479 nodes and 31,070 edges.

- **Synthetic Datasets**
  We first generate three random undirected graphs from a power-law degree distribution with exponent $\alpha = (2.0, 1.6, 1.2)$ respectively. The first, which we refer to as Synthetic 1, has $\alpha = 2.0$, 10,000 nodes, 32,799 edges, with a strongly connected component size of 0.985. The second, Synthetic 2, has $\alpha = 1.6$, 10,000 nodes, and 132,117 edges. The third, Synthetic 3, has has $\alpha = 1.2$, 3,500 nodes, and 140,592 edges.

  The node labels are generated based on structural equivalence. To do this, we first use the RolX algorithm to extract recursive features for each node. Then, we use non-negative matrix factorization to group the features into roles. We set the number of roles, a hyperparameter, to four. We chose four after plotting many histograms of the distribution of feature vector cosine similarity between a random node and other nodes in the graph, and noticing there were usually three to four peaks. Then we add a little bit of noise to the labels, controlled by a probability parameter p, so that they are not perfectly correlated to the RolX roles. This is done by uniformly sampling one percent of the nodes and randomly changing their node labels.

  Then to dive deeper into structural coefficient and the augmentation ratio, we construct a fourth synthetic dataset. This dataset is generated from a power-law degree distribution with $\alpha = 1.6$. It has 4,000 nodes and 35,560 edges. For this dataset, we introduce a parameter ρ (rho) which is similar to p above in that it randomly selects nodes to change labels. However, ρ does not randomly select a new class for a node, but instead randomly selects a neighbor of the node's class and replaces the existing label with this neighbor's. The idea is as ρ increases, the node labels become less dependent on node structure and more dependent on node homophily. This allows us to see what happens to ANN as the network becomes more homophilic and less structurally dependent.

During node2vec and GraphSAGE training, we hide the node labels for a certain proportion of the network (20% for our experiments). The hidden node labels get divided further into a validation and test set. We train the models on the nodes whose labels are not masked, and then predict the labels for the validation set nodes. Once we are satisfied with validation set performances, we predict the labels for the test set nodes.

5. Results

5.1. Node Classification

To evaluate the performance of the node classification task, we use standard classification accuracy. For GraphSAGE, the loss function we want to minimize is the negative log-likelihood loss.

In Table 1 and 2, we show our results on the three generated synthetic datasets, comparing the model performance of GraphSAGE and node2vec with and without the augmented neighborhood. In order to have comparable results, we tune the hyperparameters on normal GraphSAGE/node2vec based on a validation set, and then freeze the hyperparameter settings when switching to ANN. In other words, we do not change the hidden dimensions, regularization parameters, learning rate, etc when running GraphSAGE/node2vec with ANN after tuning on GraphSAGE/node2vec without ANN. This allows us to attribute

the differences in accuracy to the augmented neighborhood. From tables 1 and 2, there are dramatic improvements in test accuracy after augmenting the neighborhood with ANN for both GraphSAGE and node2vec

Dataset	GraphSAGE	ANN
Synthetic 1	0.791	**0.8555**
Synthetic 2	0.6495	**0.744**
Synthetic 3	0.6114	**0.7686**

Table 1. Test accuracy on Synthetic datasets for GraphSAGE and GraphSAGE-ANN

Dataset	node2vec	ANN
Synthetic 1	0.674	**0.8215**
Synthetic 2	0.6755	**0.9295**
Synthetic 3	0.7757	**0.9257**

Table 2. Test accuracy on Synthetic datasets for node2vec and node2vec-ANN

The improvements in accuracy make sense due to the nature of the synthetic datasets node label generation. Since a node's label is correlated with the structural role of the node, the model gains information by adding artificial links to structurally equivalent nodes. The augmented neighborhood provides additional structural information to each node during training. Thus, we have shown how augmenting the neighborhood can improve model performance when the node labels are based on structural equivalence.

Figure 3 shows a plot of node2vec test accuracies on the synthetic datasets. We notice that the non-augmented model does better in the early epochs, but is eventually surpassed by the augmented model. ANN surpasses the non-augmented model within 100 epochs for synthetic datasets 1 and 2. Furthermore, ANN continues to improve for more epochs than the regular node2vec. For example, synthetic 3 ANN continues improving past 350 epochs while synthetic 3 node2vec stagnates around epoch 200.

5.2. Model Configuration

For the GraphSAGE models, we use SGD optimizer with momentum parameter 0.9, learning rate 0.01 and Adam optimizer with beta = $(0.9, 0.999)$, learning rate 0.001. We also use a learning rate annealer that halves the current rate if the training loss does not improve by 1×10^{-4} in 100 epochs. Depending on the dataset, the GraphSAGE hidden dimension size is set to $32, 64, 96$, or 128 and we use 2 to 3 layers. The L2 regularization penalty and dropout also depend on the dataset, but are between $[1e-5, 5e-4]$, and $[0.25, 0.5]$ respectively. For node2vec, we use embedding dimension 128, walk length 8 to 10, positive samples context size 5, and number of negative samples 4. We train for

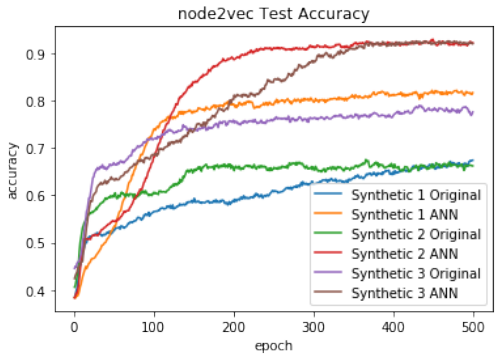

Figure 3. Plot of node2vec test accuracies with and without ANN on Synthetic datasets

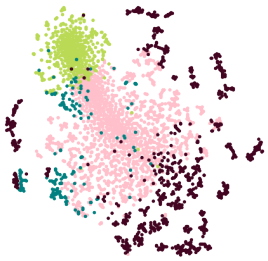

Figure 4. t-SNE Synthetic 1 dataset node2vec-ANN

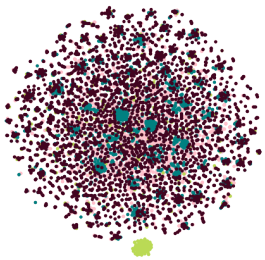

Figure 5. t-SNE Synthetic 1 dataset node2vec

500 to 700 epochs on a NVIDIA T4 GPU. If the validation set performance does not improve in 200 epochs, the model early stops.

5.3. Analysis

5.3.1 Embeddings

We use t-SNE to produce two-dimensional visualizations of the embeddings generated by the node2vec algorithm with and without ANN. The results are shown in Figures 4, 5, 6, and 7. Here, the four colors represent the four node labels. On the synthetic datasets, the embeddings are not able to separate into distinct clusters using original node2vec. The four node labels are mixed together, except for the light green cluster. On the other hand, there is a noticeable improvement when using ANN as the embeddings clearly separate into distinct groupings. This separation is ideal because we want nodes with the same classes to be close to each other in embedding space. Conversely, we want nodes with different labels to be farther away from each other, which is the case here.

5.3.2 Structural Coefficient

To demonstrate the structural coefficient, we generate random undirected graphs of size 5000 nodes from a power-law

Figure 6. t-SNE Synthetic 2 dataset node2vec-ANN

Figure 7. t-SNE Synthetic 2 dataset node2vec

degree distribution with exponents 1.2, 1.6, and 2.0. For each graph, we generate the node labels based on structural equivalence with the RolX algorithm as mentioned above in Section 4. However, this time, we vary the number of roles, and the amount of noise added to the labels. The amount of noise is controlled by a probability parameter $p \in (0, 1)$ where p is the probability of changing a node's label to a random label. From table 3, it is evident that increasing p decreases the structural coefficient. This is because as p increases, the node labels becomes less dependent on structural equivalence and more likely to be random. Thus, the structural coefficient gets weaker and strictly decreases over the range of p.

p	4 roles	6 roles	10 roles
0.001	1.653	3.005	0.9846
0.1	0.662	1.522	0.6419
0.2	0.396	0.8535	0.4178
0.4	0.1545	0.2750	0.1880
0.6	0.0739	0.0946	0.0766
0.8	0.0193	0.0246	0.0327
0.999	0.0013	0.0005	0.0012

Table 3. Simulation of structural coefficients for power-law graphs with varying number of roles

We compute the structural coefficient of several real-world graphs and the synthetic graphs. The results are displayed in Table 4. Cora and Citeseer have low structural coefficient and thus their respective node labels are not strongly related to node structure. On the other hand, BZR has a fairly high structural coefficient. For the synthetic datasets, the structural coefficients are relatively quite high due to the nature of their node label generation. We then alter synthetic datasets 1 and 2 so that their labels are instead generated by homophily using the Louvain algorithm. The structural coefficient in this case is very low as expected since the node labels are based on community and local neighborhoods instead of node roles.

Using synthetic dataset 4, we alter ρ, the probability of randomly selecting a node and replacing its node label with

Dataset	ψ
Cora	0.156
CiteSeer	0.108
BZR	0.402
Synthetic 1 - Homophily	0.018
Synthetic 2 - Homophily	0.031
Synthetic 1 - Structural	0.458
Synthetic 2 - Structural	1.428
Synthetic 3 - Structural	2.504

Table 4. Structural coefficients ψ computed on real-world graphs and synthetic graphs

ρ	ψ	Accuracy
0.001	1.412	0.798
0.1	1.064	0.719
0.2	0.837	0.654
0.4	0.367	0.608
0.7	0.066	0.548
0.999	0.004	0.553

Table 5. Simulation of GraphSAGE-ANN accuracy for various structural coefficients ψ on Synthetic Dataset 4 and fixed augmentation ratio of 1.

a neighbor's label in order to see if there is a relationship between a network's structural coefficient and the model accuracy. According to Table 5, decreasing the structural coefficient decreases the model performance of ANN. This is expected as increasing ρ makes the node labels more based on homophily instead of structural equivalence. Thus, as the structural coefficient decreases, it appears that extending a node's neighborhood based on similar structure nodes actually harms the model, infusing the model with irrelevant and/or too much information.

5.3.3 Augmentation Ratio

We conduct experiments on the ANN augmentation ratio, which is the fraction of top similar nodes in terms of structural equivalence to add proportional to a node's degree. For example, if a node has degree 4, and the augmentation ratio is 0.5, then we add the top 2 similar nodes to ANN. Increasing the augmentation ratio naturally increases the number of edges and the size of the augmented neighborhood.

From table 6, it appears that as ρ increases, the optimal augmentation ratio decreases. As the structural coefficient decreases, the node labels become more based on homophily and thus the amount of artificial edges to add should also decrease. In fact, for $\rho = 0.9$, original GraphSAGE does better than ANN; since at this point, adding edges based on structural equivalence appends information that is detrimental to the model.

ρ	ANN-0	ANN-0.25	ANN-0.5	ANN-1	ANN-2
0.01	0.683	0.708	0.759	0.745	0.773
0.1	0.649	0.676	0.678	0.719	0.696
0.2	0.637	0.644	0.654	0.654	0.645
0.5	0.541	0.543	0.559	0.543	0.545
0.9	0.578	0.519	0.523	0.521	0.522

Table 6. Simulation of GraphSAGE-ANN accuracies for various augmentation ratios and various structural coefficients on Synthetic Dataset 4. Note that ANN-0 is regular GraphSAGE.

5.3.4 Real Datasets

We ran ANN on three real-world datasets: Citeseer, Cora and BZR. For each dataset, we compared the performance with the augmented neighborhoods with the non-augmented cases. The results show that ANN performed better on the BZR dataset in terms of accuracy but performed slightly worse on CiteSeer and Cora. The performance of ANN on these real-world datasets is largely correlated with the structural coefficient of the graph, which confirms our hypothesis that ANN is more effective when there is more structural equivalence. These results also align with the synthetic dataset results we had in Section 5.3.

Dataset	Algorithm	Valid Accuracy
Cora	GraphSAGE	0.710
Cora	GraphSAGE-ANN	0.695
Cora	node2vec	0.838
Cora	node2vec-ANN	0.836
CiteSeer	GraphSAGE	0.699
CiteSeer	GraphSAGE-ANN	0.673
BZR	GraphSAGE	0.562
BZR	GraphSAGE-ANN	**0.576**

Table 7. Results on real-world graphs in terms of the dataset, algorithm, and validation set accuracy

The structural coefficient was high for BZR, so having node embeddings correlate with structural features is highly beneficial for node classification. The structural coefficient of CiteSeer and Cora are low, and thus the node labels are not strongly related with the amount of structural equivalence in the network.

We explored varying the augmentation ratio for the BZR dataset, as shown in Figure 8. This ratio represents the influence of structural equivalence vs. homophily in the context of node classification. Our experiments show that the performance of GraphSAGE-ANN is optimal with an augmentation ratio of 0.4 on BZR. We believe this ratio value is related to the degree of impact of structural features to the node label. As seen from the graph, having a higher or lower augmentation ratio would lead to a drop in performance.

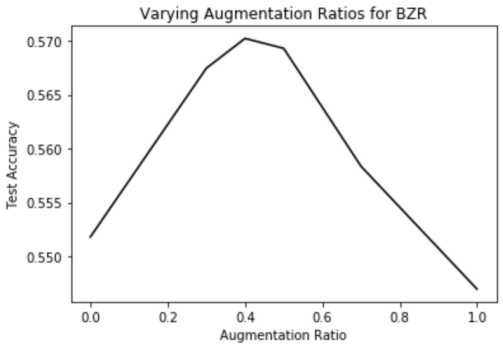

Figure 8. Plot of Augmentation Ratio vs.Test Accuracy on BZR

6. Conclusion

We described a novel extended neighborhood approach to two graph representation learning methods, GraphSAGE and node2vec. The model automatically augments a node's neighborhood to capture structural information from outside a node's local neighborhood. This leads to an improvement in node classification performance of GraphSAGE and node2vec on three synthetic random power law distribution graphs, where the labels were generated by structural equivalence. This also leads to a performance boost in the BZR dataset, where structural features play a big role in identifying the node labels.

We also proposed the structural coefficient, a new metric for measuring the structural equivalence. The structural coefficient can be thought of as a ratio of between-class variation to within-class variation. We expect the structural coefficient of a network to be high if structural roles are highly indicative towards node classification.

Future work includes experimenting with augmenting only the neighborhood of the first layer of the GraphSAGE model. The rest of the layers would then have the same edge list as the original graph. Also, we would like to utilize more features as input to the GraphSAGE and node2vec models. We also want to find more real datasets that have high structural coefficients. These would theoretically perform better under ANN, as per our simulations. Finally, we want to find the optimal augmentation ratio as a function of structural coefficient.

References

[1] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. *CoRR*, abs/1607.00653, 2016.

[2] W. L. Hamilton, R. Ying, and J. Leskovec. Inductive representation learning on large graphs. *CoRR*, abs/1706.02216, 2017.

[3] K. Henderson, B. Gallagher, T. Eliassi-Rad, H. Tong, S. Basu, L. Akoglu, D. Koutra, C. Faloutsos, and L. Li. Rolx: Structural role extraction mining in large graphs. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 08 2012.

[4] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. *CoRR*, abs/1609.02907, 2016.

[5] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad. Collective classification in network data. *AI magazine*, 2008.

[6] J. J. Sutherland, L. A. O'Brien, and D. F. Weaver. Spline-fitting with a genetic algorithm: a method for developing classification structureactivity relationships. *Journal of Chemical Information and Computer Sciences*, 43(6):1906–1915, 2003. PMID: 14632439.