AM-DGCNN: Leveraging Graph Attention Networks and Edge Attributes for Link Classification in Knowledge Graphs

Dhroov Pandey[+]
Department of Computer Science and Engineering
University of North Texas, Denton, TX, USA
dhroovpandey@my.unt.edu

Tong Shu[*]
Department of Computer Science and Engineering
University of North Texas, Denton, TX, USA
tong.shu@unt.edu

***Abstract*—Graph-based representations are increasingly popular for storing and managing information through knowledge graphs, which capture entities and their relationships. However, these knowledge graphs often suffer from incomplete link information. To address this issue, link classification methods can be used to predict and verify missing connections. Traditional methods use heuristics to guess the associativity patterns of the graph, but this results in a loss of generalizability. Recently, supervised heuristic learning methods have improved the link classification accuracy by learning the heuristic that fits best for a particular graph. Specifically, the SEAL framework [37], as a state-of-the-art supervised heuristic learning tool, excels in learning associativity patterns by analyzing local enclosing subgraphs to classify links. However, DGCNN, a graph neural network (GNN) model in this framework, lacks the capability to process edge attributes, leading to poor classification accuracy in knowledge graphs with rich link information. Hence, this paper proposes an Augmented Model of the DGCNN (AM-DGCNN) by replacing Graph Convolution Networks with Graph Attention Networks to better incorporate link information. With extensive experiments, we demonstrate that our AM-DGCNN in the SEAL framework can achieve up to 99% AUC and 97% precision for classifying links in knowledge graphs.**

***Index Terms*—Graph attention networks, edge attributes, link classification/prediction, knowledge graphs**

I. Introduction

Knowledge graphs are an efficient information representation form. They use nodes to indicate entities in a given domain and define relationships between these entities with edges/links. These knowledge graphs can be analyzed for scientific discovery based on their structure, which encodes various patterns of entity associations. Unfortunately, knowledge graphs are often incomplete to represent full knowledge. If these existing link patterns can be used to classify associations between two entities with unknown relationship, it will be more efficient for domain experts to make up missing information in knowledge graphs. Therefore, it is critical to accurately predict the association types between two entities with unknown relationship in knowledge graphs.

This is well-known as link prediction or classification problems. In essence, link classification is a more generalized problem of link prediction, because link prediction can be considered as binary link classification for existing and no relationship between two entities. Specifically, the link classification problem can be formulated as follows: Given a knowledge graph $G = (V, E)$ and a link $e = (x, y) \in E$, we aim to classify the link e by determining the nature of the relationship between these entities x and y.

Most of traditional link classification methods [3], [12] use a graph structure-based heuristic (e.g., Adamic-Adar [2], preferential attachment [17], common neighbors) to extract topological graph features and translate them into scores, which are used to judge the association between any two nodes. If the heuristic explores the target node's neighborhoods within h hops, it is called h-order heuristic. Higher-order heuristics, such as PageRank [5] and SimRank [10], usually outperform lower order heuristics, such as common neighbors, and thus are more widely adopted. However, the performance of graph structure-based heuristics depends on whether representative topological features can be extracted as a good indicator of node associativity. Unfortunately, this is not guaranteed. Furthermore, the link classification heuristics only use the structural features and don't take explicit node features into account.

A more effective methodology used by link prediction is *supervised heuristic learning* [36], [37], which refers to learning the effective heuristics via machine learning. This is usually done by processing the adjacency matrix of the graph to generate a graph embedding that can be fed to a deep neural network (DNN). However, this is often computationally infeasible for very large graphs, such as biological knowledge graphs. To address this issue, the existing methods usually extract and process an enclosing subgraph, such as the union of k-hop neighborhoods of the target nodes. This seems to lose high-order heuristic features. Zhang et al. discovered a theory that any high-order heuristic, if it is γ-decaying, can be approximated by a low-order neighborhood [37]. Hence,

This research work is done in the Smart High-performance and Ubiquitous Systems (SHU'S) lab at the University of North Texas.

+ Dhroov Pandey (ORCID: 009-0006-7107-6231) is an undergraduate student in Computer Science at the University of North Texas.

* Corresponding author: Tong Shu, Assistant Professor, Department of Computer Science and Engineering, University of North Texas, Denton, TX 76207, USA (ORCID: 0000-0001-8617-1772)

these enclosing subgraphs can be used to calculate all the low-order heuristics and sufficiently approximate high-order heuristics. Therefore, the Weisfeiler-Lehman Neural Machine (WLNM) [36] uses DNNs to process enclosing subgraphs of the target nodes and achieve a good performance. Further, Zhang et al. upgraded WLNM in the SEAL framework [37] by replacing the DNN with a standard dynamic graph convolutional neural network (DGCNN) architecture [34], one type of graph neural networks (GNNs) [13]. However, this architecture fails to take into consideration link features, which are very important for link classification.

In this paper, we proposed an Augmented Model of DGCNN (AM-DGCNN), which incorporates both node and link feature processing into the GNN architecture design and outperforms vanilla DGCNN in the SEAL framework. Also, we auto-tuned the hyperparameters of the AM-DGCNN using DeepHyper [1] for the optimal performance. Extensive experimental results over realistic knowledge graphs show that AM-DGCNN can predict link classes with up to 98% accuracy.

II. Background

A. Graph Neural Networks

Similar to specialized neural architectures utilized for image classification, GNNs specialize in processing graph-structured data and generate embeddings. They usually accept the adjacency matrix, a node attribute matrix, and sometimes edge attributes as inputs. GNNs extract node features from a graph via a message passing scheme that progressively transforms the graph attributes (e.g., node attribute vectors and edge attributes) into representations that encapsulate the context of the attribute in the graph. These representations can then be aggregated by a graph aggregation layer [38] to generate a graph representation vector which can be processed by regular DNNs. GNNs are able to identify the associativity patterns of various diverse networks and make them highly generalizable for network analysis. Furthermore, GNNs can generate associativity functions of a much broader range than known heuristics.

Graph Convolution Networks (GCNs) [13]: Kipf et al. applied convolution operations to transform node features and follow a neighborhood aggregation scheme to propagate information across the graph. The aggregation is usually achieved by performing matrix multiplication between the adjacency matrix, the node information matrix, and an intermediate weight matrix that performs the convolution operation. GCNs have achieved good performance for node classification in balanced graphs [11], [35].

Graph Attention Networks (GATs) [33]: Velickovic et al. applied the principle of attention [4] to GNNs. A single graph attention layer takes as input a set of node attribute vectors, and applies a transformation to map those vectors to a new feature space. Then, self-attention [32] is applied to each node to determine the importance of the neighboring nodes' features. The attention coefficients are shared by all the nodes in the network. This allows the model to give selective preference to more relevant nodes in a node's neighborhood as opposed to GCN, which gives the entire neighborhood equal importance. Furthermore, GAT is able to process edge attribute vectors and utilizes them while calculating attention coefficients, hence, incorporating link information into node transformations.

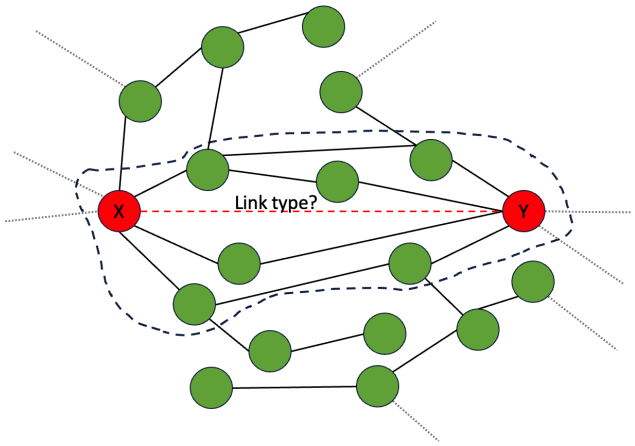


Fig. 1: Local neighborhood intersection subgraph

B. SEAL

We adapt a link prediction framework, SEAL [37], for our link classification problem. Our underlying intuition is that link prediction can be viewed as a binary classification or existence classification problem. Therefore, SEAL can be extended to other forms of link classification. This method extracts local enclosing subgraphs around the target nodes. Then, a node information matrix is constructed using node attribute vectors composed of the following three parts: (i) a double-radius node labeling (DRNL), (ii) a node embedding generated by node2vec [8] and (iii) explicit node features. The subgraph adjacency matrix along with the information matrix is processed by a GNN to output classification results. Since SEAL is a GNN-agnostic framework, Wang et al. applied DGCNN as a GNN model in [34].

Theory of γ-decaying heuristics: In [37], it is shown that if a high-order heuristic is a γ-decaying heuristic [37], it can be learnt from a k-hop enclosing subgraph, but the improvement of approximation error decreases at least exponentially with k. Also, it shows that popular high-order heuristics, such as Katz Index [16], PageRank, and SimRank, can be classified as γ-decaying heuristics. This has an intuitive justification, since it is reasonable that nodes far away from the target nodes in the graph would have very little effect on their associativity patterns. Since processing an entire knowledge graph with a huge number of nodes and a high degree density is computationally expensive, we use local sub-graphs enclosing the two target nodes, which include sufficient information for classifying the link between the target nodes.

Double-Radius Node Labeling: Zhang et al. assign a non-negative integer to each node as a label based on their distances from the target nodes [37]. Specifically, let a and b be the two target nodes; n indicates the node to be labeled. Then, the label of node n is defined as the value of the symmetric function $D(x,y) = 1+\min(x,y)+(x+y)[(x+y)/2+(x+y)\%2-1]/2$,

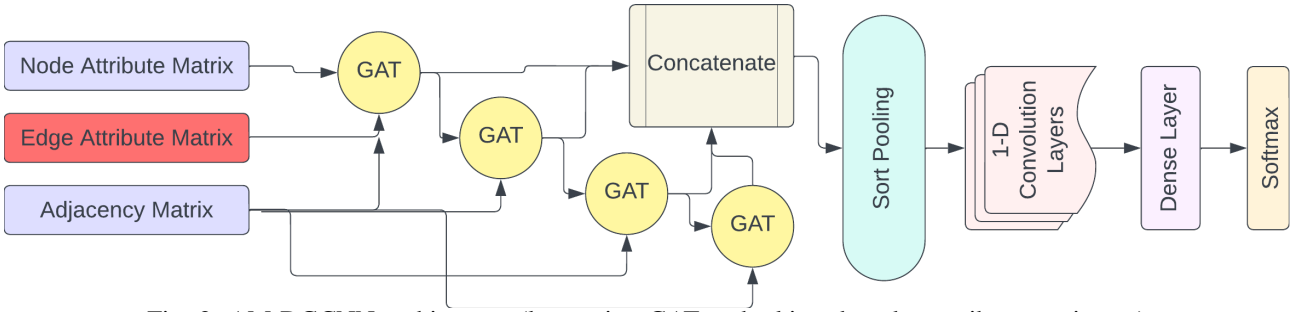


Fig. 2: AM-DGCNN architecture (leveraging GAT and taking the edge attributes as inputs)

where $x \in \mathbb{N}$ is the shortest distance between n and a, and $y \in \mathbb{N}$ is the shortest distance between n and b. The target nodes are given the distinctive label "1". If the node is not reachable by any one of the target nodes, it is given the null label "0". The DRNL label is appended to the node feature vector to encode additional topological information.

III. Methodology

Based on the SEAL framework, our approach extracts a local subgraph around the target nodes. Then, we generate node and edge attribute matrices for the nodes and edges in the subgraph. Next, the subgraph along with the attribute matrices is processed by a GNN to generate a vector representation that is enhanced by convolution and pooling operations and passed through a dense layer for classification.

A. Subgraph Extraction

To extract the local subgraph, we consider the k-hop neighborhoods of the target nodes and use the intersection of these neighborhoods as the induced subgraph, as shown in Fig. 1. Specifically, we use the intersection for the PrimeKG graph (See Section IV), instead of a union operation, to reduce the subgraph size, which has been verified empirically; however, we use the union for the other knowledge graphs. This subgraph contains all the nodes that lie on paths between two target nodes with the length less than $2k$. We chose $k = 2$ for the following two reasons: 1) The target nodes (e.g., drug/disease) with high degrees have many neighborhoods within k hops, so that the enclosing subgraphs are too big to be processed for large k. 2) A low order local subgraph can be used to approximate the high order heuristics with sufficient accuracy as mentioned in [37].

B. Node and Edge Attribute Matrix Generation

Our node attribute vector is the concatenation of two vectors. The first is a one hot encoding of the node type in a knowledge graph. The second encapsulates structural information as a one hot encoding of its DRNL [37]. We empirically observed that Node2Vec embeddings did not enhance prediction accuracy for knowledge graphs, such as PrimeKG. Therefore, we ignore it for faster training and inference. To generate the edge attribute matrix, we compressed the 30 relationships with two link types: encoding positive or negative interaction. The edge attribute vectors are two-dimensional one-hot encoding vectors with the encoding classes: positive link and negative link. In a attribute matrix, each row represents the feature vectors of one node or edge and each column represents a feature.

C. AM-DGCNN Architecture

We improve the original GNN link prediction architecture (i.e., vanilla DGCNN [34]) in the SEAL framework. The DGCNN uses multiple graph convolution layers sequentially to generate node embeddings. Each subsequent layer is able to aggregate more neighborhoods than the previous one. The node embeddings from all the graph convolution layers are concatenated to form the final node embeddings. All the final node embeddings are aggregated by a sort pooling [38] to generate an embedding for the entire graph. This graph embedding is processed by convolution and pooling layers and then fed into a dense classifier (i.e., a deep neural network) for subsequent feature processing.

While the DGCNN is a very powerful architecture, it has a serious shortcoming for link classification: it is unable to process link attribute information, which is vital for processing knowledge graphs. Therefore, in order to incorporate edge features, we augment the DGCNN architecture by modifying its message passing layers, i.e., replacing the GCNs with GATs. The AM-DGCNN architecture is visualized by Fig. 2.

D. Hyperparameter Auto-Tuning

The prediction accuracy of the well-trained AM-DGCNN relies on its hyperparameter settings in the training process. To further enhance the performance of AM-DGCNN, we auto-tune the hyperparameters of the AM-DGCNN, listed in Tbl. I, using DeepHyper [1], which is a Python library to automate machine learning processes, such as hyperparameter tuning and neural architecture search. In DeepHyper, hyperparameter auto-tuning requires to define the search space of the hyperparameter configurations and an evaluator function that is used to indicate the prediction accuracy over different hyperparameter configurations. The search strategy is an optimization algorithm for exploring the search space and identifying the best hyperparameter configuration according to evaluator function values. Here, we selected the Centralized Bayesian Optimization search strategy due to its computational efficiency for large search spaces.

TABLE I: Hyperparameters of GNNs and their options

HyperParameters	Options
Learning Rate	[0.000001, 0.01]
GNN Layer (GAT/GCN) Hidden Dimensions	16, 32, 64, 128
Sort Aggregator k Value	$5, 6, \cdots, 150$

IV. Datasets

To test the generalizability of our methodology, we evaluate the augmented model of DGCNN using three different datasets of knowledge graphs with link features: PrimeKG [7], OGBL-BioKG [9], and WordNet-18 [6], [15]. Also, we use the Cora graph dataset without link features in the Planetoid repositor as a benchmark to explain that the superiority of our AM-DGCNN comes from its capacity of distinguishing link features.

PrimeKG is a multi-modal knowledge graph with 129,375 nodes encompassing 10 major biological scales: proteins/-genes, drugs, diseases, phenotypes, exposures, anatomical regions, pathways, biological processes, cellular components and molecular functions. It is designed to support precision medicine analysis and has a dense collection of relationships between nodes with 4,050,249 links with 30 relationships encoding "positive" and "negative" interactions. In our analysis, we are going to be looking at the drug-disease links which can be classified as "Indication" (i.e., positive link), "Off-label use" (i.e., positive support link), and "Contra-indication" (i.e., negative link). PrimeKG contains 7,957 drug nodes and 17,080 disease nodes with 42,631 drug-disease interactions, providing sufficiently large training and test data. We used 6000 training links and 2000 testing links.

OGBL-BioKG is a biological knowledge graph with five node types and 51 relationships/edge types for ample features. It is a large graph with about 100,000 nodes and around 4,000,000 edges. In the prediction task, we intend to classify the link between two protein nodes into seven possible relationships/link types. The bottleneck of the graph's performance is the limited number of data samples in the target category. We used 1300 training links and 200 testing links.

WordNet-18 contains word senses and relates them to lexical meaning and is commonly used for text analysis. This knowledge graph contains 40943 nodes and about 150,000 edges. It has a homogeneous node topology (i.e., only one type of nodes) and 18 edge classes. This dataset allows us to test the impact of edge features in the absence of any node features by measuring the GAT's capability to extract information from only link information. The vanilla DGCNN should not be able to learn much meaningful information from the WordNet. We used 13000 training links and 4000 testing links.

The **Cora** dataset in the **Planetoid** repository is a well-known link prediction benchmarking dataset. It is a citation network with 2708 nodes in seven classes and 5429 edges of uniform topology (i.e., only one edge type). We use this dataset to measure the performance of our AM-DGCNN against vanilla DGCNN in the scenarios where edge features cannot be exploited. In essence, this benchmark directly compares the node feature message passing scheme of GAT and GCN. We divided the links into an 80-20 train-test split.

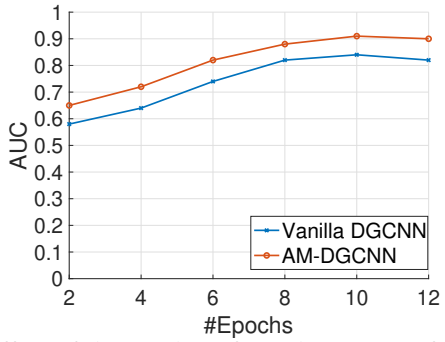


Fig. 3: Effect of the number of epochs on AUC for Cora with auto-tuned hyperparameters

Table II summarizes the benchmark datasets.

TABLE II: Summary of Datasets

Dataset	#Node types	#Edge types	#Nodes	#Edges
PrimeKG	10	30	129,375	4,050,249
OGBL-BioKG	5	51	100k	4,000,000
WordNet-18	1	18	40,943	150k
Cora	7	1	2708	5429

V. Evaluation

In this section, we compared the prediction accuracy of our AM-DGCNN with the vanilla DGCNN over aforementioned four knowledge graph datasets.

A. Metrics

To measure the prediction accuracy of GNN models for link classification, we use the following two metrics:

1) **Area Under the ROC Curve (AUC)**: A receiver operating characteristic (ROC) curve is a graph showing the performance of a classification model at all classification thresholds by plotting the true positive rate (TPR) versus the false positive rate (FPR) at different classification thresholds. Here,

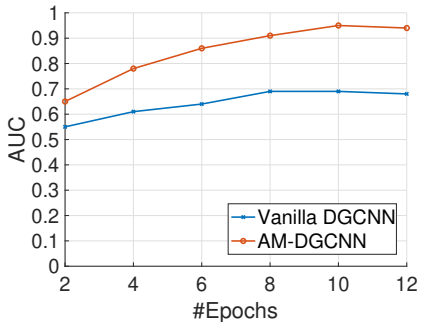
- TPR (i.e., recall) is $TP/(TP + FN)$, where TP and FN are the numbers of true positives and false negatives, respectively.
- FPR is $FP/(FP+TN)$, where FP and TN are the numbers of false positives and true negatives, respectively.

AUC (between 0 and 1) measures the entire two-dimensional area underneath the entire ROC curve (i.e., integral calculus) from (0,0) to (1,1). "1" indicates the best prediction accuracy. To plot the AUC, we randomly choose one class from all the classes as the positive class and treat the rest of classes as negative classes.

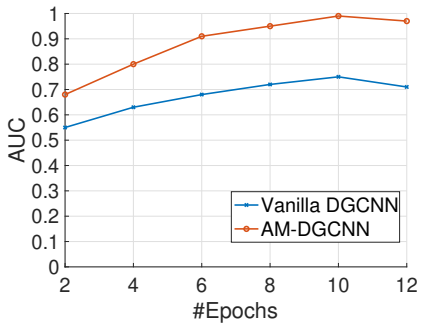
2) **Average Precision (AP)**: Precision is $TP/(TP + FP)$. To calculate the precision of each class, we treat the observed class as the positive class and the rest classes as negative classes. Based on this, AP is the mean of precision values for all the classes.

B. Experiment Design

We conducted the following two sets of experiments: (i) We auto-tuned the hyperparameters of AM-DGCNN and vanilla

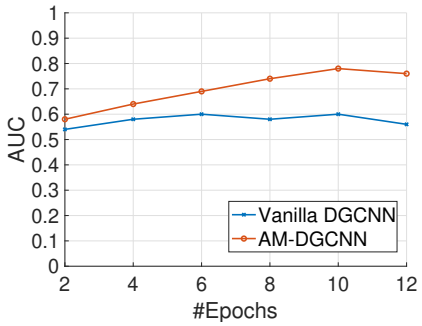


(a) Default hyperparameters

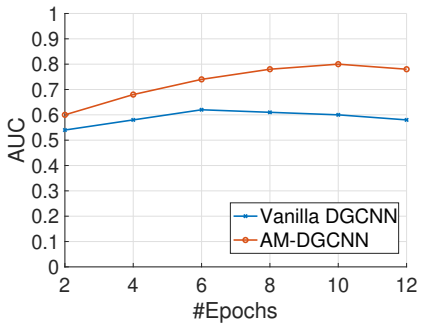


(b) Auto-tuned hyperparameters

Fig. 4: Effect of the number of epochs on AUC for PrimeKG

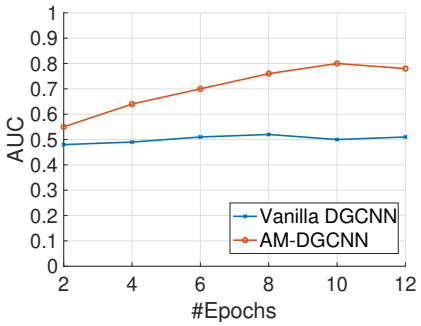


(a) Default hyperparameters

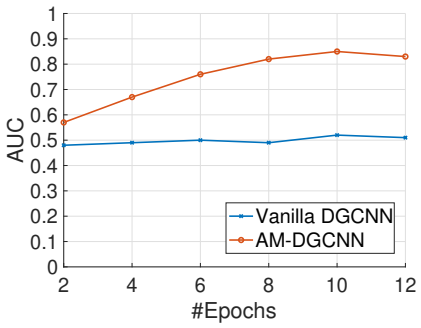


(b) Auto-tuned hyperparameters

Fig. 5: Effect of the number of epochs on AUC for OGBL-BioKG



(a) Default hyperparameters



(b) Auto-tuned hyperparameters

Fig. 6: Effect of the number of epochs on AUC for WordNet-18

TABLE III: Prediction accuracy of different GNNs

Dataset	AM-DGCNN		Vanilla DGCNN	
	AUC	AP	AUC	AP
PrimeKG	0.99	97%	0.75	55%
OGBL-BioKG	0.80	75%	0.66	40%
WordNet-18	0.85	89%	0.52	38%
Cora in Planetoid	0.91	92%	0.84	88%

DGCNN for the Cora dataset to identify the optimal hyperparameter configurations, and evaluated their performance with the same hyperparameter configurations for the other three knowledge graphs with link features (i.e., PrimeKG, OGBL-BioKG, and WordNet-18). (ii) We auto-tuned the hyperparameters of AM-DGCNN and vanilla DGCNN for each of three graphs with link features, respectively, and evaluated their performance with the optimal hyperparameter configurations in each scenario.

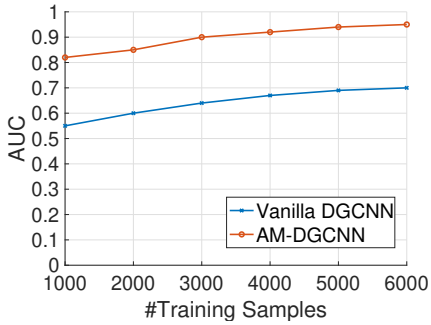
The reasons for such experimental design is that the Cora dataset, due to its lack of edge attributes, would tune the models solely for node attribute-based message passing, hence maximizing the vanilla DGCNN's performance capability while not optimizing the GAT processing on edge attributes (i.e., link features). Whereas in the second set of experiments, the edge attribute processing would be fully tuned and maximize its contribution. This allows us to observe the effectiveness of edge attributes in a two-fold way: (i) the general performance improvement resulting from incorporating edge features into GNN models when ignoring the effect of dedicated hyperparameter settings, and (ii) the maximum performance improvement resulting from edge feature processing in GNN when taking hyperparameter optimization potential into account.

C. Results on Prediction Accuracy

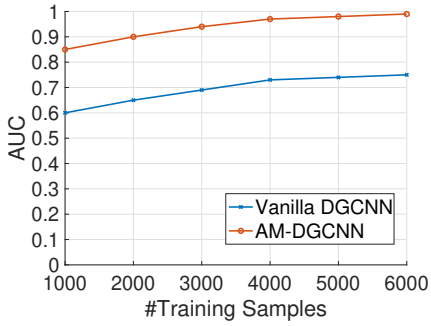
To compare the prediction accuracy of AM-DGCNN and vanilla DGCNN, we list the AUC and AP of these two models well-trained through entire training data in the second set of experiment in Tbl. III, which shows that the AM-DGCNN outperforms vanilla DGCNN for all the datasets. Despite the lack of edge classes in the Cora dataset, attention-based graph convolutions is still superior to standard GCN. Particularly, AM-DGCNN has the most significant performance improvement over vanilla DGCNN for the WordNet-18 dataset without explicit node features. Since it is empirically observed that the adjacency matrix does not contain sufficient features, the important information to be encoded can only depends on the edge features (less possibly topology). As a result, the vanilla DGCNN performs like a random guesser when it fails to learn any associativity patterns from pure geometrical topology. Therefore, by effectively making use of edge attributes, AM-DGCNN elevates its performance over the vanilla DGCNN.

D. Results on Time-Efficiency of Model Training

To compare the training time-efficiency of AM-DGCNN and vanilla DGCNN, we measured AUC of these two models after the different numbers of training epochs from 2 to 12 at the interval of 2 over all the training samples of each dataset, and plot the measurements in Figs. 3–6. These figures shows that the AUC of the AM-DGCNN is consistly higher than

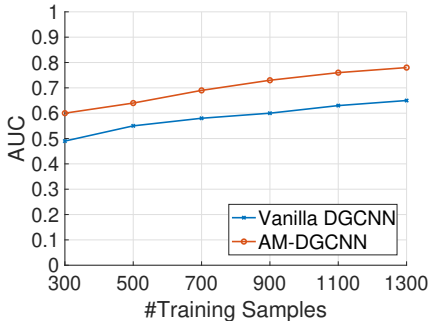


(a) Default hyperparameters

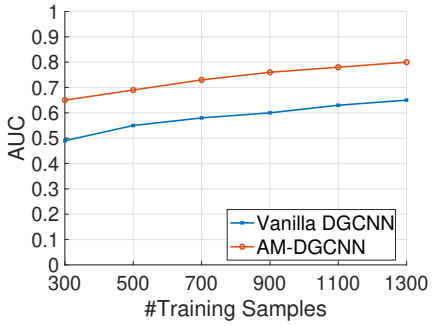


(b) Auto-tuned hyperparameters

Fig. 7: Effect of the number of training samples on AUC for PrimeKG

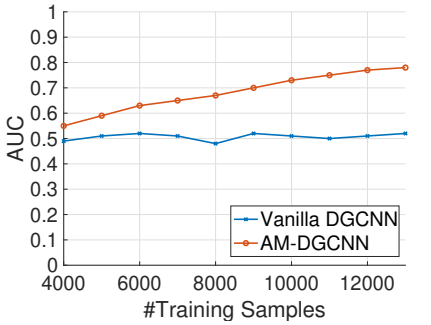


(a) Default hyperparameters

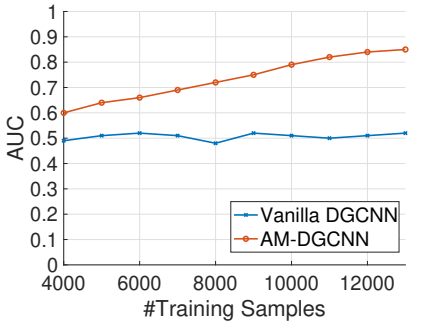


(b) Auto-tuned hyperparameters

Fig. 8: Effect of the number of training samples on AUC for OGBL-BioKG



(a) Default hyperparameters



(b) Auto-tuned hyperparameters

Fig. 9: Effect of the number of training samples on AUC for WordNet-18

that of vanilla DGCNN during the entire training process, which validates the superior time-efficiency of AM-DGCNN training. Also, we discovered that these two models perform best just after 10 training eporchs. Thus, we can conclude that the AM-DGCNN obtains performance gains in terms of prediction accuracy without sacrificing speed of learning.

E. Results on Data-Efficiency of Model Training

To evaluate the training data-efficiency of these two models, we measured the AUC of the well-trained models (after 10 eporchs) based on different number of training samples in each dataset, and plot the measurements in Figs. 7–9. From these figures, we can see that the AUC of the AM-DGCNN is always greater than that of vanilla DGCNN for all the numbers of training samples, which shows the better data-efficiency of AM-DGCNN training. Particularly, the AUC of AM-DGCNN is beyond 0.9 with a half of training samples from PrimeKG, because PrimeKG has sufficient data samples and richer explicit node information. Also, the AUC of AM-DGCNN can still reach 0.8 with around 2/3 of training samples from OGBL-BioKG, which has a very limited total number of training samples (no more than 1100), and WordNet-18, which lacks node features. Therefore, the data-efficiency of AM-DGCNN is much better than vanilla DGCNN.

F. Results on Hyperparameter Sensitivity Analysis

In general, the prediction accuracy of a machine learning model is affected by its hyperparameter settings, but hyperparameter tuning involves an extremely high training cost. Therefore, it is important to analyze the effect of hyperparameter configurations on the performance of our AM-DGCNN. In view of this, we plot the AUC of AM-DGCNN and vanilla DGCNN with default hyperparameters and auto-tuned hyperparameters in Figs. 4–9. Here, default hyperparameters are hyperparameters auto-tuned for the Cora dataset without edge features. By comparing each pair of figures (a) and (b) in Figs. 4–9, we observed that the AM-DGCNN's performance gains are insensitive to hyperparameter configurations. The AM-DGCNN outperforms the vanilla DGCNN with similar margins in the default and auto-tuned hyperparameter settings. Therefore, the superiority of AM-DGCNN over vanilla DGCNN is very stable. In addition, the cost of hyperparameter auto-tuning is savable for AM-DGCNN, even through it is computationally expensive.

VI. Related Work

Existing link classification methods can be classified into two categories: heuristics based approaches and supervised heuristic learning approaches.

A. Heuristic-based Approaches

A common methodology for link classification is utilizing network heuristics. There are several heuristic methods to translate topological node features to scores and associate them with nodes in a graph. These scores can then be utilized by classifiers to learn a threshold value for separating classes. The order of a heuristic is defined as the radius of the neighborhood that the heuristic processes. A k-order

heuristic processes the k-hop neighborhood of the node being scored. Generally, high-order heuristics outperform low-order ones. Katragadda et al. use Common Neighbor, Adamic-Adar Index [2], Jaccard coefficient, preferential attachment [17], and edge weight in heuristics to extract features, which are then processed by a decision tree-based classifier for edge classification [12]. In [31], Vasavada et al. utilize Common Neighbor, Adamic Adar, Node Degree, and PageRank [5] in topological heuristics to generate features. Then, these features along with explicit node attributes are taken as inputs by classifiers based on Logistic Regression, Feed-forward Neural Network, and Long Short term Memory Network (LSTM) [30]. While these classifiers are able to learn from these features, (similar to the paradigm in image classification) a better approach is to let the model learn to extract features from the network context (e.g., topology and attributes).

The existing heuristics-based approach has the following three issues.

Generalizability: Arbitrary given heuristic is universally inappropriate. In other words, even if a heuristic performs well for a certain graph structure and domain, it would not necessarily perform well for a different structure and domain. For instance, the common neighbor heuristic is very good at predicting links in social networks, because a person is highly likely to have a relationship with the friends of his friends in this domain. However, it has contra-positive results in protein-protein interaction (PPI) networks because any two proteins having a common interactive protein makes it less possible for them to interact [37].

Theoretical Limits: Various complex networks in unexplored domains might have associativity patterns that are not encoded into any known heuristics. Furthermore, there might be unknown heuristics that can outperform the existing methodologies of network analysis in a domain. Hence, heuristic methods are limited by theoretically discovered topological features of a graph.

Node Feature Utilization: In the majority of knowledge networks, feature vectors associated with nodes can be incorporated into network analysis. However, several heuristic methods only consider the structural graph features and disregard the node features.

B. Supervised Heuristic Learning Approaches

Weisfeiler-Lehman Neural Machine (WLNM) [36] is an early supervised graph heuristic learning methodology. The methodology involves extracting the union of the k-hop neighborhoods of the target nodes. The vertices are then relabeled and reordered according to the Weisfeiler-Lehman algorithm [36] to encode structural information. The updated adjacency matrix is processed by a DNN to learn graph patterns. This approach, however, has several drawbacks. First, it is implicitly feeding heuristics to its DNN by re-labeling the nodes via a topological scheme instead of letting the DNN learn the topology heuristics entirely on its own. Second, the adjacency matrix for the graph needs to be of a fixed size because a feed-forward network can only process tensors with a fixed size. Hence, the enclosing subgraphs have to be truncated or padded with zeros, resulting in a loss or dilution of structural features. Moreover, the WLNM cannot learn from explicit node features due to limitations of the adjacency matrix representation and hence, misses out on important graph attribute information.

VII. Conclusion

In this work, we enhanced the GNN architecture in the SEAL framework and propose AM-DGCNN by modifying the DGCNN architecture to incorporate link information into the graph embedding process. Also, we adapted the SEAL framework to make it fit for our AM-DGCNN. We found that attention-based node processing outperforms GCNs and edge features significantly boost the GNN's performance without a significant cost to computational latency. The AM-DGCNN architecture was able to obtain better prediction accuracy than the vanilla DGCNN model in all the cases of our experiments. The performance difference become more substantial when datasets included edge features. We conclude that AM-DGCNN is a suitable model for link classification in knowledge graphs.

To further validate the advantage of leveraging GATs and edge attributes for link classification, we will apply the AM-DGCNN to predict data transfer in extensive graph-based scenarios beyond knowledge graphs, such as distributed neural network models [14], [18], directed acyclic graph (DAG)-structured workflows [19], [20], [24], [26]–[28], high-performance networks [25], [29], and wireless mesh networks [21]–[23], in the future.

Acknowledgment

This research is sponsored by National Science Foundation under Grant No. OAC-2306184 with the University of North Texas.

References

[1] DeepHyper: A python package for scalable neural architecture and hyperparameter search, 2018. https://github.com/deephyper/deephyper.

[2] L. A. Adamic and E. Adar. Friends and neighbors on the web. Social Networks, 25(3):211–230, 2003.

[3] C. Aggarwal, G. He, and P. Zhao. Edge classification in networks. In 2016 IEEE 32nd International Conference on Data Engineering (ICDE), pages 1038–1049, 2016.

[4] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate, 2016.

[5] M. Bianchini, M. Gori, and F. Scarselli. Inside pagerank. ACM Trans. Internet Technol., 5(1):92–128, feb 2005.

[6] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko. Translating embeddings for modeling multi-relational data. In Proc. of Conf. on Neural Information Processing Systems (NeurIPS), volume 26, 2013.

[7] P. Chandak, K. Huang, and M. Zitnik. Building a knowledge graph to enable precision medicine. Scientific Data, 10(1):67, 2023.

[8] A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In Proc. of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), page 855–864, New York, NY, USA, 2016. Association for Computing Machinery.

[9] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec. Open graph benchmark: datasets for machine learning on graphs. In Proc. of Intl. Conf. on Neural Information Processing Systems (NeurIPS), Red Hook, NY, USA, 2020.

[10] G. Jeh and J. Widom. Simrank: a measure of structural-context similarity. In Proc. of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), page 538–543, New York, NY, USA, 2002.

[11] Q. Jiao, Y. Jin, and Y. Liu. Node classification without features using graph convolutional network. In IEEE Intl. Conf. on Information Technology, Big Data and Artificial Intelligence (ICIBA), volume 2, pages 991–994, 2021.

[12] S. Katragadda, H. Karnati, M. Pusala, V. Raghavan, and R. Benton. Detecting adverse drug effects using link classification on twitter data. In IEEE Intl. Conf. on Bioinformatics and Biomedicine (BIBM), pages 675–679, 2015.

[13] T. N. Kipf and M. Welling. Semi-supervised classification with graph convolutional networks, 2017. https://arxiv.org/abs/1609.02907.

[14] T. Kundu and T. Shu. HIOS: Hierarchical inter-operator scheduler for real-time inference of dag-structured deep learning models on multiple gpus. In Proc. of the 25th IEEE International Conference on Cluster Computing (Cluster), pages 95–106, Santa Fe, NM, USA, Nov 2023.

[15] G. A. Miller. WordNet: A lexical database for english. Communications of the ACM, 38(11):39–41, Nov. 1995.

[16] E. Nathan and D. A. Bader. A dynamic algorithm for updating katz centrality in graphs. In Proc. of IEEE/ACM Intl. Conf. on Advances in Social Networks Analysis and Mining (ASONAM), page 149–154, New York, NY, USA, 2017.

[17] M. E. J. Newman. Clustering and preferential attachment in growing networks. Physical Review E, 64:025102, Jul 2001.

[18] D. Pandey, J. Ghebremichael, Z. Qi, and T. Shu. A comparative survey: Reusing small pre-trained models for efficient large model training. In Proc. of Workshops of IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC-W), Atlanta, GA, USA, Nov 2024.

[19] T. Shu, Y. Guo, J. Wozniak, X. Ding, I. Foster, and T. Kurc. Bootstrapping in-situ workflow auto-tuning via combining performance models of component applications. In Proc. of ACM/IEEE International Conference for High Performance Computing, Networking, Storage, and Analysis (SC), pages 1–15, St. Louis, MO, USA, Nov 2021.

[20] T. Shu, Y. Guo, J. Wozniak, X. Ding, I. Foster, and T. Kurc. POSTER: In-situ workflow auto-tuning through combining component models. In Proc. of the 26th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP), pages 467–468, Seoul, South Korea, Feb-Mar 2021.

[21] T. Shu, M. Liu, and Z. Li. Spectrum allocation for distributed throughput maximization under secondary interference constraints in wireless mesh networks. In Proc. of the 20th IEEE International Conference on Computer Communications and Networks (ICCCN), Maui, HI, USA, Aug 2011.

[22] T. Shu, M. Liu, Z. Li, and Q. Wu. Interference pair-based distributed spectrum allocation in wireless mesh networks with frequency-agile radios. In Proc. of the 8th IEEE International Conference on Sensing, Communication, and Networking (SECON), pages 82–90, Salt Lake City, UT, USA, Jun 2011.

[23] T. Shu, M. Liu, Z. Li, and A. Zhou. Joint variable width spectrum allocation and link scheduling for wireless mesh networks. In Proc. of IEEE International Conference on Communications (ICC), Cape Town, South Africa, May 2010.

[24] T. Shu and C. Q. Wu. Energy-efficient mapping of big data workflows under deadline constraints. In Proc. of the 11th Workshop on Workflows in Support of Large-Scale Science (WORKS) in conjunction with ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis (SC), pages 34–43, Salt Lake City, UT, USA, Nov 2016. http://ceur-ws.org/Vol-1800/paper5.pdf.

[25] T. Shu and C. Q. Wu. Bandwidth scheduling for energy-efficiency in high-performance networks. IEEE Transactions on Communications, 65(8):3359–3373, 2017.

[26] T. Shu and C. Q. Wu. Energy-efficient dynamic scheduling of deadline-constrained MapReduce workflows. In Proc. of the 13th IEEE International Conference on eScience (e-Science), pages 393–402, Auckland, New Zealand, Oct 2017.

[27] T. Shu and C. Q. Wu. Performance optimization of Hadoop workflows in public clouds through adaptive task partitioning. In Proc. of IEEE International Conference on Computer Communications (INFOCOM), pages 2349–2357, Atlanta, GA, USA, May 2017.

[28] T. Shu and C. Q. Wu. Energy-efficient mapping of large-scale workflows under deadline constraints in big data computing systems. Elsevier Future Generation Computer Systems, 110:515–530, 2020.

[29] T. Shu, C. Q. Wu, and D. Yun. Advance bandwidth reservation for energy efficiency in high-performance networks. In Proc. of the 38th IEEE Conference on Local Computer Networks (LCN), pages 541–548, Sydney, Australia, Oct 2013.

[30] R. C. Staudemeyer and E. R. Morris. Understanding LSTM—a tutorial into long short-term memory recurrent neural networks, 2019. https://arxiv.org/pdf/1909.09586.

[31] V. Vasavada and W. Wang. Link prediction and edge classification for orphan disease concept graph, 2019. https://snap.stanford.edu/class/cs224w-2019/project/26418782.pdf.

[32] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Łukasz Kaiser, and I. Polosukhin. Attention is all you need. In Proc. of Conf. on Neural Information Processing Systems (NeurIPS), pages 5998–6008, Long Beach, CA, USA, Dec 2017.

[33] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. Graph attention networks, 2018. https://arxiv.org/abs/1710.10903.

[34] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. Dynamic graph CNN for learning on point clouds. ACM Transactions on Graphics (TOG), 38(5), Oct 2019.

[35] B.-Y. Xiao, C.-C. Tseng, and S.-L. Lee. Node classification using graph convolutional network with dropout regularization. In IEEE Region 10 Conference (TENCON), pages 84–87, Dec 2021.

[36] M. Zhang and Y. Chen. Weisfeiler-lehman neural machine for link prediction. In Proc. of the 23rd ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining, pages 575–583. ACM, 2017.

[37] M. Zhang and Y. Chen. Link prediction based on graph neural networks. In Proc. of Conf. on Neural Information Processing Systems (NeurIPS)), pages 5165–5175, 2018.

[38] M. Zhang, Z. Cui, M. Neumann, and Y. Chen. An end-to-end deep learning architecture for graph classification. Proc. of the AAAI Conference on Artificial Intelligence, 32(1), Apr 2018.

ARTIFACT DESCRIPTION/EVALUATION APPENDIX

A. Summary of the Experiments Reported

1) Abstract: We provide this artifact appendix to make our results reproducible. Our main results are *Model AUC* and *Model Precision*. We used the metrics subpackage from the torch library to compute our benchmarks.

2) Artifacts: source code GitHub Link

git clone https://github.com/SHUs-Lab/MLGHPCE24DP.git

3) Relevant Hardware Details: The experiments were performed on **NVIDIA A40** GPU with **CUDA 11.8**

4) Applications and Versions:

- Python 3.9

5) Libraries and Versions:

- Pytorch 2.1.2+cu118
- Pytorch geometric 2.4.0
- numpy 1.24.1

B. Evaluation Experiments

1) Experiment 1: For recording the model performance across epochs, do the following:

Cora: Edit the file 'Planetoid_bench.ipynb' by modifying line 1 of block 11 to adjust the number of epochs.

OGBL-BioKG: Edit the file 'bio_kg_bench.ipynb' by modifying line 1 of block 20 to adjust the number of epochs.

WN18: Edit the file 'WN18_benchmark.ipynb' by modifying line 1 of block 11 to adjust the number of epochs.

PrimeKG: Edit the file 'GNN.ipynb' by modifying line 1 of block 7 to adjust the number of epochs.

2) Experiment 2: For recording model performance across the number of training samples, do the following:

OGBL-BioKG: Edit the file 'bio_kg_bench.ipynb' by modifying line 6 of block 19 to adjust the number of batches to train.

WN18: Edit the file 'WN18_benchmark.ipynb' by modifying lines 6, 8 and 10 of block 6 to adjust the number of training samples in the trainer.

PrimeKG: Edit the file 'GNN.ipynb' by modifying line 3 of block 7 to adjust the number of batches to train.