Reasoning over Incomplete Knowledge Graph via Graph Structure Learning, Group 3

Ayushi Agarwal
ayushi15@g.ucla.edu
University of California, Los Angeles (UCLA)

Shardul Shailendra Parab
shardulparab@g.ucla.edu
University of California, Los Angeles (UCLA)

Parth Shettiwar(Research Credits)
parthshettiwar@g.ucla.edu
University of California, Los Angeles (UCLA)

Harini Suresh
sharini16@g.ucla.edu
University of California, Los Angeles (UCLA)

Anubhav Mittal
anubhavm@g.ucla.edu
University of California, Los Angeles (UCLA)

ABSTRACT

As we all know, Graph neural networks(GNNs) will be able to work the best when the entire graph structure is provided. However, it is not practical in a real-world setting to have the entire graph structure provided. One of the earlier solutions proposed to this problem is that we devise a task-specific latent structure and apply a GNN to the inferred graph. One of the major possibilities of the space of the graph is that it grows super-exponentially with the number of nodes and so the proposed structure for task-specific will be inefficient for learning the structure and GNN parameters. Another problem that arises when dealing with large graph structures is they are often incomplete. To deal with this, methods have proposed using similarity graphs based on the initial features or learning the graph structure and the latent representations simultaneously, with the latter method often achieving better results. Through this work we aim to propose a solution for completing the graph structure, leveraging transformers attention technique to learn better representations of the relations in graph, and iteratively building them upon the original graph. In end, we show some preliminary analysis as part of this project which motivated us to pursue this line of direction.

KEYWORDS

Graph Neural Networks, K hop augmentation, Attention, Incomplete Graphs, Structural Learning

ACM Reference Format:
Ayushi Agarwal, Shardul Shailendra Parab, Parth Shettiwar(Research Credits), Harini Suresh, and Anubhav Mittal. 2022. Reasoning over Incomplete Knowledge Graph via Graph Structure Learning, Group 3. In *Proceedings of ACM Conference (CS 249).* ACM, New York, NY, USA, 8 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Graphs are ubiquitous in representing objects and their complex interactions. Several constructs inherently involve the notion of graphs, such as social networks, molecular structures, knowledge bases, recommendation systems, etc. The ability to exploit relationships amongst entities in the data is a crucial one for advancing the state of artificial intelligence.

GNNs generate node embeddings by recursively aggregating data from neighboring nodes. This iterative approach has cascade effects, since little noise is propagated to neighbors, decreasing the quality of many others' representations. Consider a social network, in which nodes represent users and edges indicate friendship connections. Because of the recursive aggregation nature of GNNs, fraudulent accounts may easily perform adversarial attacks, making account credibility difficult to predict. Recent research shows that undetected, intentional modifications in graph topology can easily result in inaccurate predictions for the majority of GNNs. As a result, high-quality graph structure is typically necessary for GNNs to acquire effective representations.

However with data, the graph structures have gone exponentially large, leading to inefficient learning since most of the graph structures are sparse. Due to this incompleteness prevelent in graph structures, it becomes utmost important to learn meaningful representations of the entities to predict the relations between far nodes of the graph. There are also other issues associated with such incomplete graphs such as an edge starvation problem, and some of the recent work has focused on dealing with them.

This motivates us to solve the problem of graph structure learning (GSL) [13]. GSL methods aim to learn the graph structure and GNN parameters jointly. Since real-life graphs have multiple relations between the entities, we plan to solve this in heterogeneous graph setup. GSL approaches for adaptively learning graph structures for GNNs have recently been presented, the majority of which parameterize the adjacency matrix and optimize it together with the GNN parameters. However, these algorithms are all designed for homogeneous networks and cannot be easily used to relation graphs due to the following issues: (1) The heterogeneity of relation graphs When learning a homogeneous network with only one type of relation, we usually only need to parameterize one adjacency matrix. A relation graph, on the other hand, is composed of several relations, each representing a distinct component of the relation network. Because managing these many relationships consistently will inevitably restrict the ability of graph structure learning. Managing this diversity is a challenging task. (2) The complex links seen in relation graphs. As a result of intricate interactions between various relations and node properties, many forms of underlying network structure emerge.

Keeping in mind the complexity associated with this problem statement, we restrict our search space only over k-Hop neighbours [10] in our algorithm, while enforcing edge masking technique to avoid over-fitting over the incomplete small graphs. As a future work attention [2] can also be used for efficient graph generation [5]. Overall the main contributions through this work are:

- Identify the issue of GNNs inability to accurately learn the entities representation in incomplete graphs by evaluating on WN18RR dataset.
- Leverage transforms attention technique to learn better nodes representation using CompGCN model in a relation graph setup and show its efficacy using empirical results on FB15K-237 and WN18RR datasets.
- Formulate the idea of graph structure learning and propose an algorithm to solve this problem, using positive and negative edge sampling to effectively learn the graph structure over many iterations.

2 PROBLEM DEFINITION AND FORMALIZATION

Graph neural networks (GNNs) work well when the graph structure is provided. However, this structure may not always be available in real-world applications. One solution to this problem is to infer a task-specific latent structure and then apply a GNN to the inferred graph. Unfortunately, the space of possible graph structures grows super-exponentially with the number of nodes and so the task specific supervision may be insufficient for learning both the structure and the GNN parameters.

Several works have been done in the past to solve noisy graphs, edge starvation problem, however we believe state-of-the-art model is the first to come now [3].

Thus, we device the problem into two parts as follows:

Link prediction using Attention in Relational Graphs

We solved the problem of Link Prediction [12] using CompGCN [9] that is work done on Relational Graphs. We try to adopt the attention framework used in Heterogeneous Graph Transformers [4] and see how adding the attention in models such as CompGCN affect the prediction results. These will be the preliminary results for our further investigation on Incomplete Graphs.

Reasoning over Incomplete Graphs

Second, we try to see how dropping edge from the graphs (Taking all the nodes but $n\%$ of relations) affect the learning. We make these analysis in Shallow and Deep Embedding Models. For our purposes we use TransE [1] as Shallow Embeddings and CompGCN, our attention based relation graph learning model (CompGCN + att) are affected by dropping of edges.

Based on our analysis we finally formulate a new problem statement of Graph Neural Network and suggest methods to provide the solution to this problem.

Our problem statement is as follows. *In real world applications, we are never sure that a complete graph is provided. Empirically proven, this incompleteness of Graph affects Deep Embeddings for Knowledge Graphs than Shallow embeddings. Thus, given a graph G, we want to find a Model M such that the if x% of G is dropped, M will complete G and give results for downstream tasks such as Link Prediction comparable to the results when the complete G is provided.*

3 RELATED WORK

The proposed algorithm has drawn motivation from a variety of previous works involving the predictions in a multi-relational graph, applying attention based mechanisms and our novelty in the algorithm proposed to solve incomplete graphs.

1. **TransE:**
   TransE [1] is a very simple method which models relationships by interpreting them as translations operating on the low-dimensional embeddings of the entities. More specifically, for each triplet $(\mathbf{s}, \mathbf{r}, \mathbf{o})$, it aims to find fix-sized embeddings for both entities and relations such that the translation $\mathbf{s} + \mathbf{r}$ is close to $\mathbf{o}$. This relation is imposed by minimizing the $l2$-distance metric between the two $||\mathbf{s} + \mathbf{r} - \mathbf{o}||$ across the training examples.
   To make the model more robust, they use negative sampling based on the training triplets - for a given training triplet $(\mathbf{s}, \mathbf{r}, \mathbf{o})$ and entity/node set E, they sample from the following:
   $$S'_{(\mathbf{s},\mathbf{r},\mathbf{o})} = \left\{ (\mathbf{s'}, \mathbf{r}, \mathbf{o}) \mid s' \in E \right\} \cup \left\{ (\mathbf{s}, \mathbf{r}, \mathbf{o'}) \mid o' \in E \right\}$$
2. **R-GCN**:
   R-GCN [7] were one of the first models to show that the GCN framework can be applied to modeling relational data, specifically to link prediction and entity classification tasks. Their link prediction model can be regarded as an autoencoder consisting of (1) an encoder: an R-GCN producing latent feature representations of entities, and (2) a decoder: a tensor factorization model exploiting these representations. For the encoder part, they define weight matrices and normalization constants for each relation and a fixed size embedding for representing each entity and apply the standard GCN message-passing framework on them. For reducing model size, they use parameter sharing - they define these weight matrices as sum of basis matrices which are learned from the data.
   For the decoder part, they use DisMult [11] to combine the subject and object nodes and find a prob. vector representing the probabilities of the link between them belonging to different classes.
3. **CompGCN:**
   CompGCN [9] is a Graph Convolutional framework which jointly embeds both nodes and relations in a relational graph. It leverages the entity-composition operations from a variety

of entity-relation composition operations from Knowledge Graph Embedding techniques. It also has the potential to scale with the number of relations. The novelty of CompGCN is it learns a d-dimensional representation along with the node embeddings. In order to solve the problem of over-parameterization, relations are represented by vectors while applying GCNs on relational graphs. The update equation can be written as in equation 1.

$$\boldsymbol{h}_v = f\left(\sum_{(u,r)\in\mathcal{N}(v)} \boldsymbol{W}_r \boldsymbol{h}_u\right) \tag{1}$$

On elaborating the equation above, N(v) is a set of immediate neighbours of v for its outgoing edges. The above step will be affected by over-parameterization, hence in CompGCN, we apply composition of a neighboring node u with respect to its relation 'r'. This leads the update equation to be re-written as in equation 2.

$$\boldsymbol{h}_v = f\left(\sum_{(u,r)\in\mathcal{N}(v)} \boldsymbol{W}_{\lambda(r)} \phi\left(\boldsymbol{x}_u, \boldsymbol{z}_r\right)\right) \tag{2}$$

CompGCN differs from Relational GCN in a way that the edge embeddings are not used in the computation of the update step in learning the relations. CompGCN performs the following tasks:

- Link Prediction can be defined to predict the missing facts based on the available facts in the Knowledge Graphs.
- Node Classification can be defined as predicting the labels of the nodes based on the features of the node and graph structure.
- Graph Classification involves the prediction of the label of the graph. We would be learning the representation of the structure of the graph which will be fed into the classifier.

(4) **SLAPS: Self-Supervision Improves Structure Learning for Graph Neural Networks**: This is a method that provides more supervision for inferring a graph structure through self-supervision techniques. This model learns the parameters of a graph neural network and a graph structure of nodes simultaneously based on how the nodes are connected. SLAPS is mainly responsible for solving the edge starvation problem.
The architecture of this approach is given in Figure 3.
The overall algorithm of SLAPS is as follows:

- The generator receives the node features and produces a non-normalized, non-symmetric adjacency having both positive and negative values.
- The adjacency processor is responsible for making the values symmetric and normalized.
- The adjacency processor makes the values positive, symmetries and normalizes The node classes will be predicted using node features based on resulting adjacency At the bottom, some noise is added to the node features.
- The GNN(DAE) model will use the noisy features and generated adjacency to predict the features.

(5) **Heterogenous Graph Transformer**:
This architecture focuses on modeling Web-scale heterogeneous graphs [4]. To model heterogeneity, the node and edge type dependent parameters are used to characterize the attention over each edge, hence trying to maintain different representations for different types of nodes and edges. For handling the dynamic heterogeneous graphs, a relative temporal encoding technique has been integrated into HGT to capture the dynamic structural dependency.

(6) **ConvE**:
ConvE is a multi-layer convolutional architecture for link prediction: it is defined by a single convolution layer, a projection layer to the embedding dimension, and an inner product layer. It takes in the subject **s** and **o** embeddings as input and predicts the prob. of a relation **r** between them.

Taking inspiration from the above resources, we propose a novel methodology for incomplete graph generation.

4 DATASETS

We propose to use attention based modelling for CompGCN [9] and incomplete graph generation using Label Propogation as downstream tasks. Hence, we use the following datasets.

- **WN18RR Dataset:** is a link prediction dataset created from WN18, which is a subset of WordNet[6]. WN18 consists of 18 relations and 40,943 entities. However, many text triples are obtained by inverting triples from the training set. Thus the WN18RR dataset is created to ensure that the evaluation dataset does not have inverse relation test leakage. WN18RR dataset contains 93,003 triples with 40,943 entities and 11 relation types.
- **FB15k-237 Dataset:** FB15K-237 is a variant of the Fb15K Freebase dataset where inverse relations are removed. Fb15K has a total of 592,213 triplets with 14,951 entities and 1,345 relationships. FB15K dataset suffered from major test leakage through inverse relations, where a large number of test triples could be obtained by inverting triples in the training set.

5 METHODS DESCRIPTION

5.1 CompGCN+ATT (Novelty for Relational Graphs)

First we tried to integrate embeddings to Relational datasets using the architecture of Heterogeneous Graph Transformers. This was done to set the premises of how attention can improve learning on simple models and as thus can be used as problem solution to Incomplete Graph Generation.

Since, in these experiments we just use Relational Graphs as our datasets, we try to model attention framework on CompGCN. Thus we define the framework as follows. Let n_r be the number of relations. We define weights for every relation as $\mathbf{W_{attr}} \in R^{n_r \times d}$, where d is the hidden dimension. Since, the number of relations in Relational Datasets is very large we deal with the problem of over-parameterization using bases as used in RGCN [7]. We say that all the relations can be modelled as the linear combination of basis vectors. So we define $\mathbf{c} \in R^{n_r \times n_b}$ and $\mathbf{e} \in R^{n_b \times d}$

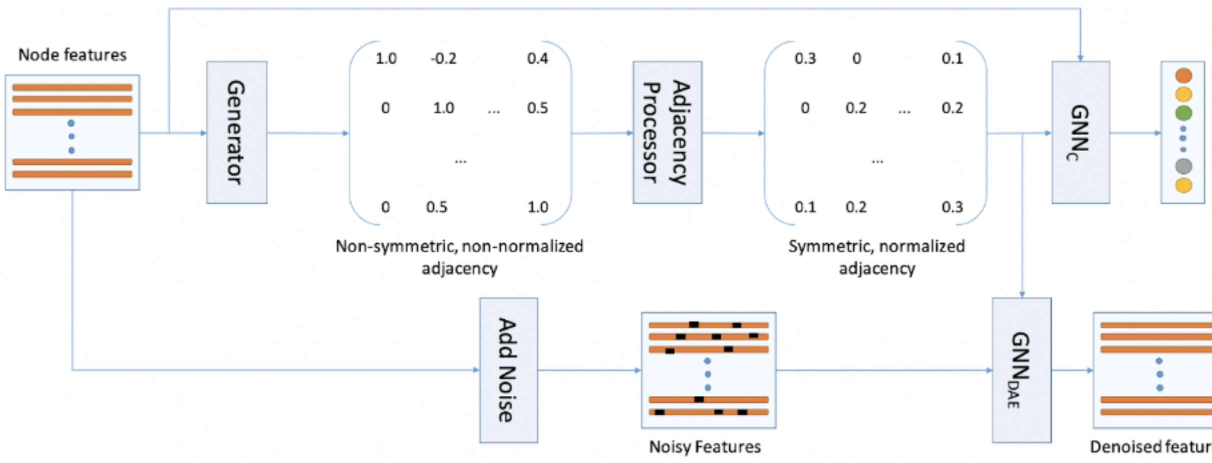

Figure 1: Architecture of SLAPS

where n_b defines the number of bases, e are basis vectors and c are the coefficients to model the relation weights.

Thus for each edge e attention is calculated as follows.

$$k(s) = \mathbf{W_k} \cdot \mathbf{x_v}(s)$$
$$q(t) = \mathbf{W_q} \cdot \mathbf{x_v}(t)$$
$$v(s) = \mathbf{W_v} \cdot \mathbf{x_v}(s)$$
$$a(s,e,t) = k(s)\mathbf{W_{attr}}^{(e)} q(s)$$

Where k, q and v are query vectors for the edges source and target nodes, s and t are source and target nodes and x_v are node embeddings. a(s, e, t) is the attention value for the edge.

The message passing takes place as follows. Here, $N(t)$ are the neighbours of the target node, h_v are the hidden state node embeddings and α is the learnable parameter to give weights for the residual connection.

$$att = \mathbf{Softmax}_{\forall s \in N(t)}(a(s,e,t))$$
$$\mathbf{h} = \sum att \cdot v(s)$$
$$\mathbf{h_v} = \alpha \mathbf{x_v}(\mathbf{t}) + (1-\alpha)\mathbf{h}$$

The architecture for integrating attention with CompGCN is given in Figure 2

5.2 Incomplete Graph Generation

We proposed the following methodology for our work. As part of our structure learning setup, we will perform edge masking for positive edge samples and negative k-hop edge sampling to iteratively learn the graph structure and learn the GNN optimal weights.

(1) **Edge masking**: For every epoch, we mask some edges, based on sampling techniques explained later and use the rest of graph for the complete epoch. The masked edges are labelled as unknown relation and considered as positive samples, for which loss is computed accordingly based on the learning from remaining graph. Note, when the edges are masked we allow message passing through these edges but using the unknown relation label. This unknown relation is simply denoted by *<UKN>* while training.

(2) **K-Hop neighbour edges augmentation**: After performing edge masking, we find k-hop neighbours of all the nodes and create a k-hop neighbour graph. Since this consists of huge number of edges, we use the sampling techniques discussed later, to create a sparser k-hop graph, and augment the edge masked graph with this k-hop graph, with *<UKN>* as the label for all of these negative edges.

Based on these above techniques, we propose our Graph Structure Learning Strategy as follows:

Initially, consider we take a graph *G(V, E)* i.e. the 100 percent graph. This graph has all the nodes and edges intact and this is the graph G that we aim to build using the Graph Structure learning paradigm. We use the concept of *IEP* i.e. the Initial Edge Percent (by default we take it as 10 percent) and we decide to keep only the IEP percentage of edges and drop or remove the remaining edges of the graph. Hence, now we get the “Starting Graph” from where Graph Structure Learning will begin. We denote this graph by $G'(V, E')$. Carefully, have a look that the number of vertices or nodes do still stay the same, hence the notation *V*.

Now, in order to learn the graph, first we perform edge masking(default value is 30 percent). As discussed earlier, edge masking means that currently we will remove the relation from a few edges (existing in the current graph *G'*) and currently keep it as unknown. These edges will be denoted by *posE* i.e. positive sample edges which have unknown relation. Additionally, using the ‘k-hop neighbors’ technique, we sample negative edges for k-hop value 2. We denote these edges as *negE* i.e. negative sample edges. All the remaining edges will be denoted by E*. Hence, the new graph created by this

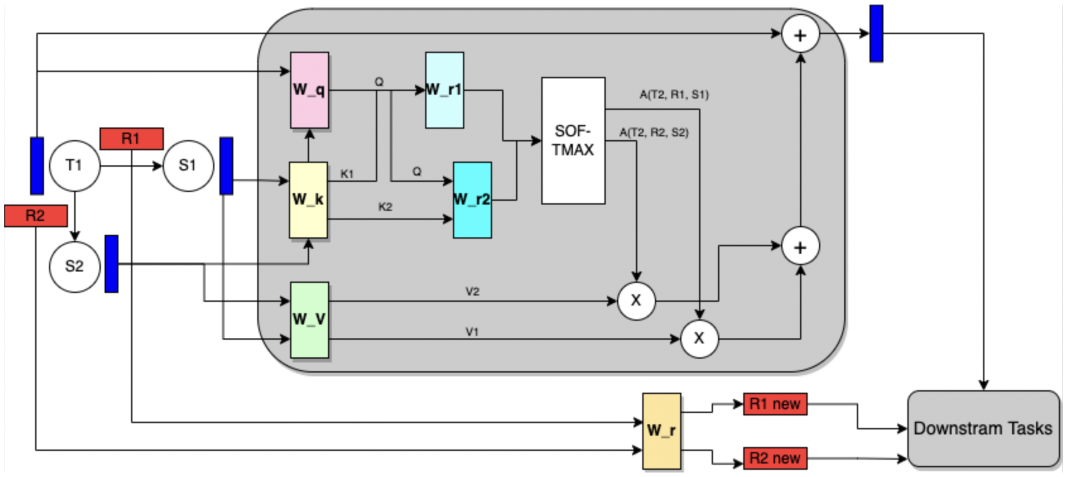

Figure 2: Architecture of CompGCN integrated with attention

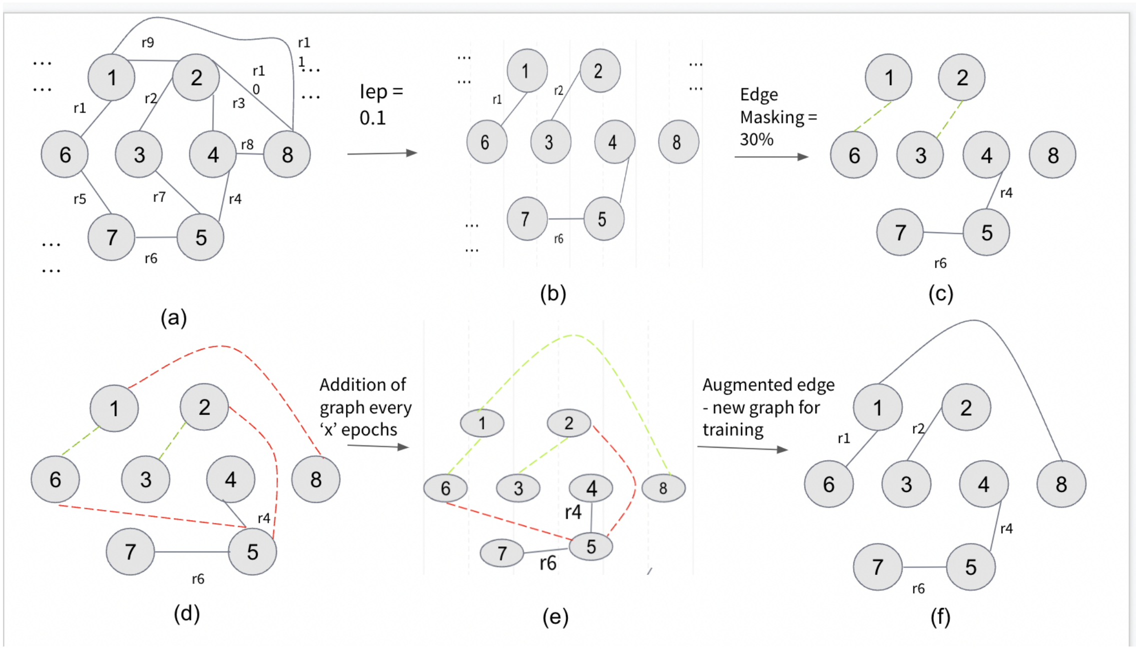
Iep = 0.1
Edge Masking = 30%
Addition of graph every 'x' epochs
Augmented edge - new graph for training
(a)
(b)
(c)
(d)
(e)
(f)

Figure 3: HGSL algorithm summarised: (a) Initial graph with all nodes and vertices *G(V, E)* (b) The reduce graph after dropping edges with an initial edge percentage(IEP) called $G'(V,E')$ (c) Perform edge masking and create positive sample edges posE (marked in green) (d) Find k-hop neighbors and choose a subset of it - negE (marked in red) and then merge all of them together $G''(V,E'')$ (e) Perform training and then add the new edge to the graph if the heuristic value if above a threshold(red edge converted to green) (f) Take the new graph G*(V, $E^\wedge$) (only the black and green edges) as the starting point for the next iteration.

exercise is $G''(V,E'')$, wherein E'' is the union of *posE*, *negE* and E^*. In practice, we do not take all the k-hop neighbors as negative samples but only a portion of them. In our case, we take the number of negative samples as the minimum of (10 times the number of positive samples, total number of k-hop neighbors).

The graph G" is then used in the training logic, wherein a particular loss objective is used(discussed further in later sections). Post this, after a particular number of iterations a new graph G*(V, $E^\wedge$) is created by adding a few edges from negE(negE*), which pass a particular scoring threshold. Herein, $E^\wedge$ is the union of E' and negE* (or to put it simply as the union of E*, posE and negE*). This new graph G* is then passed through the entire logic once again until we reach a satisfiable amount of graph completion. We call this algorithm relation graph structure learning (RGSL). Algorithm 1 and Figure 3 formulate this.

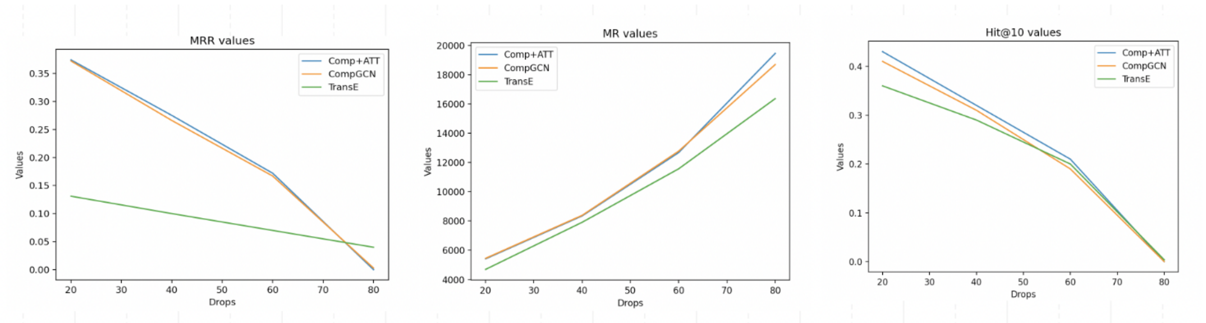

Figure 4: Results of dropping edges in the main Graph in WN18RR dataset on different models. The x axis denotes the percentage of edges dropped in the graph.

Algorithm 1 RGSL

```
Require: Sample technqiue SAMP(Graph, Edge-percentage), IEP, Masking-percent, k-hop, K, G(V, E) Edge-masking(Graph, masking-percent, model, Threshold $T$

  $G'(V,E') \leftarrow$ SAMP($G(V, E)$, IEP)

  while $N \leq$num-epochs do
      $posE \leftarrow$ Edge-masking($G'(V,E')$ , masking-percent)
      $negE \leftarrow$ k-hop($G'(V,E')$ , K)
      $G''(V,E'') \leftarrow$ merge(posE, negE, $E^*$)

      for (subject, object) in ( $G''$) do :

        pred-r $\leftarrow$ model( $G''$, $subject, object$)
        Compute Loss (pred-r,ground-truth-r) and backpropagate
    end for
    for (subject, object) in ( negE) do:

        pred-r = $\leftarrow$ model( $G''$, $subject, object$)
        s = Compute-score(pred-r)

        if s$\geq$ $t$ then :G*=$G''$ + pred-r
            end for
```

Deciding which edges to drop
We explore 2 sampling technqiues which will help us to limit our search space while performing structure learning, namely:

(1) **Uniform sampling**: Given a graph, we sample edges uniformly from it, to create a smaller sparser graph.
(2) **Neighbourhood edge sampling**: In this, we start by sampling based on nodes degrees, and as we sample edges, we increase the sampling probability of edges close to the edges already sampled.

Negative sampling procedure
Now, for incorporating the negative samples in our loss, we propose two different loss formulations depending on the type of negative examples we sample for a given positive triplet.

(1) **Relation-aware negative sampling** - For a given positive triplet $(\mathbf{s}, \mathbf{r}, \mathbf{o})$, we create negative samples $(\mathbf{s}, \mathbf{r}, \mathbf{o'})$ where $\mathbf{o'}$ is sampled from $K - Hop(\mathbf{s})$. For the decoder of form $f(\ .\ ,\ .\ )$ which takes in the input s and o and outputs a relation probability vector, our negative sample loss is

$$Loss = -\log(1 - f(\mathbf{s}, \mathbf{o'})[\mathbf{r}]) \tag{3}$$

This makes sense because

$$\begin{aligned}&\text{Minimizing } -\log(1 - f(\mathbf{s}, \mathbf{o'})[\mathbf{r}])\\ =\ &\text{Maximizing } \log(1 - f(\mathbf{s}, \mathbf{o'})[\mathbf{r}])\\ =\ &\text{Maximizing } (1 - f(\mathbf{s}, \mathbf{o'})[\mathbf{r}])\\ =\ &\text{Minimizing } f(\mathbf{s}, \mathbf{o'})[\mathbf{r}]\end{aligned}$$

which means when we try to minimize the loss, we are effectively trying to minimize the probability of that sample being true.

(2) **Relation-unaware negative sampling** - An issue with the previous loss is that minimizing the prob. for a particular relation inadvertently increases the probability for other relations. Instead, what we are aiming for here is that for a pair of entities s and r, if there exists no relation between them, then all the probabilities should be very low. To impose this, we propose another formulation of the negative sampling procedure. For a given triplet ($\mathbf{s}$, $\mathbf{r}$, $\mathbf{o}$), we create negative samples ($\mathbf{s}$, $\mathbf{o'}$) where $\mathbf{o'}$ is sampled from $K - Hop(s)$. Note that we discard the relation $\mathbf{r}$ from our negative samples here.
For the model $f(\ .\ ,\ .\ )$ taking in inputs $\mathbf{s}$ and $\mathbf{o}$ and outputs a relation probability vector. We want all the prob. values in the vector to be below a certain threshold (hyperparameter **T**), and we penalize those which exceed that value - loss function is $([f\ (\mathbf{s}\ , \mathbf{o}\ ) - \mathbf{T}]_{+})^2$

6 EVALUATION

The evaluation is done on the basis the following metrics:

(1) **Mean Reciprocal Rank(MRR)**:average of the reciprocal ranks of results for a sample of queries Q.
(2) **Mean Rank(MR):** average of the ranks of results for a sample of queries Q.

(3) **Hits Ratio at n(HR@n)**: a way of calculating how many "hits" you have in an n-sized list of ranked items.

7 EXPERIMENTS AND RESULTS

7.1 Experimental setup

Baselines
We wanted to evaluate two things - impact of adding attention on the general performance of CompGCN, and how the performance of our CompGCN-ATT model compares to the other popular deep and shallow models for incomplete graphs. Because of this, we chose CompGCN (deep) and TransE (shallow) as baselines to compare against our model.

Implementation
All our models use an embedding dimension of size $d = 200$ for both entities and relations. Most of the model-specific parameters are found through a combination of grid-search and intuition.

For implementing the transE model, we use the embedding layer provided by pytorch. We use a constant *step size* $lr = 0.001$ - we observe that the training loss keeps going down after a point with no improvement in validation score - comfirming that we already start overfitting on our model after that point so we don't need to reduce the learning rate further. We generate negative samples randomly based on the positive samples using the standard methodology described in the original paper, and train for a batch size of 64 for all our experiments on transE. We save our model whenever we reach a current best highest val. score, and use early-stopping by keeping a *patience* variable which tracks if we have not improved the val. score after an epoch - if we haven't improved the score after 20 epochs, we stop the training process and use the model with the highest val. score as our final model for measuring test performance.

For implementation of the CompGCN and CompGCN+Att models optimizer AdamW with oneCycleLR[8] scheduler. the starting learning rate was 0.001 The model was run for maximum of 500 epochs with early stopping using validation set. The best epoch model was used for testing the model using which the results were obtained.

7.2 Results and discussion

Effect of Attention on performance
First, we compare the performance of our proposed CompGCN+ATT model to the baselines on the complete WN18RR (Wordnet) and FB15K-237 (Freebase) datasets - results can be seen in tables. Our model performs consistently a little better across all metrics compared to the other two for both datasets. One interesting thing to observe here is the jump in performance for Freebase is more pronounced - this is primarily due to the $20x$ more relations in Freebase compared to Wordnet. This is expected behaviour - more relations imply more richer weight matrices $\mathbf{W_{attr}}$ used for obtaining the attention weights, and therefore leading to better results. The results are given in Table 1 and 2

Effect of Incomplete Graphs on Shallow and Deep Embeddings

Metrics	CompGCN+ATT	CompGCN base
MRR	**0.35651**	0.34708
MR	206.62807	188.16542
HIT@10	**0.53745**	0.5248
HIT@3	**0.39177**	0.38012
Hit@1	**0.2651**	0.2585

Table 1: Results of CompGCN and CompGCN + ATT Link prediction Task on FB15K-237 dataset

Metrics	CompGCN+ATT	CompGCN base	TransE
MRR	**0.46416**	0.46219	0.14916
MR	**3078.06541**	3445.87524	3280.14885
HIT@10	**0.53925**	0.53143	0.41449
HIT@3	**0.48038**	0.47687	0.21235
Hit@1	**0.42565**	0.42581	0.01851

Table 2: Results of CompGCN + ATT, CompGCN and TransE Link prediction Task on WN18RR dataset

We compare the results of dropping the edges on Shallow embeddings (TransE) and Deep Embeddings (CompGCN, CompGCN+ATT) on different percentages of graph. The results are shown in Figure 4. We can see that deeper models easily outperform shallow models on a larger dataset, but their performance also deteriorates very steeply as we remove more edges in comparison to Shallow models. This might be because the GNN models rely on the network structure where as shallow embeddings do not have message passing from neighbours. As we drop more edges the the graph loses it's structure, thus Deep Embeddings perform worser.

8 CONCLUSION AND FUTURE SCOPE

In this paper, we proposed the use of interesting attention techniques with CompGCN type of architectures. We conduct comprehensive experiments on the FB15K-237 and WN18RR datasets and show that our proposed model outperforms the CompGCN baseline. We also looked at how the dropping of edges in the main graph is detrimental in comparison to shallow embedding techniques and the performance improves once the graph is closer to completion. Taking this as our main inspiration, we model a new Graph Structure Learning technique called the Relation Graph Structure Learning (RGSL) algorithm and also propose different loss objectives which can help learn the structure better.

The future scope for this project is up to imagination as it aims to tackle a frontier which can have a lot of applications and subdomains. A few areas under consideration are as follows: (a) Ways to formulate objectives in a better manner i.e. taking inspriation from Computer Vision tasks such as YOLO and introduce an 'edgeness' score which also tells the probability of an edge actually being an edge along with predicting a relation. (b) Scaling up the problem to large scale graphs (c) modelling for heterogenous and dynamic graphs i.e. taking the problem statement closer to real-world challenges.

ACKNOWLEDGMENTS

We are thankful to Ziniu Hu and Professor Yizhou Sun for supporting us and guiding us for the entire project.

REFERENCES

[1] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. *Advances in neural information processing systems* 26 (2013).

[2] Sanxing Chen, Xiaodong Liu, Jianfeng Gao, Jian Jiao, Ruofei Zhang, and Yangfeng Ji. 2020. Hitter: Hierarchical transformers for knowledge graph embeddings. *arXiv preprint arXiv:2008.12813* (2020).

[3] Bahare Fatemi, Layla El Asri, and Seyed Mehran Kazemi. 2021. SLAPS: Self-Supervision Improves Structure Learning for Graph Neural Networks. *Advances in Neural Information Processing Systems* 34 (2021).

[4] Ziniu Hu, Yuxiao Dong, Kuansan Wang, and Yizhou Sun. 2020. Heterogeneous graph transformer. In *Proceedings of The Web Conference 2020*. 2704–2710.

[5] Renjie Liao, Yujia Li, Yang Song, Shenlong Wang, Will Hamilton, David K Duvenaud, Raquel Urtasun, and Richard Zemel. 2019. Efficient graph generation with graph recurrent attention networks. *Advances in Neural Information Processing Systems* 32 (2019).

[6] George A Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine J Miller. 1990. Introduction to WordNet: An on-line lexical database. *International journal of lexicography* 3, 4 (1990), 235–244.

[7] Michael Schlichtkrull, Thomas N Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2018. Modeling relational data with graph convolutional networks. In *European semantic web conference*. Springer, 593–607.

[8] Leslie N. Smith. 2018. A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay. https://doi.org/10.48550/ARXIV.1803.09820

[9] Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha Talukdar. 2019. Composition-based multi-relational graph convolutional networks. *arXiv preprint arXiv:1911.03082* (2019).

[10] Guangtao Wang, Rex Ying, Jing Huang, and Jure Leskovec. 2020. Multi-hop attention graph neural network. *arXiv preprint arXiv:2009.14332* (2020).

[11] Bishan Yang, Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. 2014. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575* (2014).

[12] Muhan Zhang and Yixin Chen. 2018. Link prediction based on graph neural networks. *Advances in neural information processing systems* 31 (2018).

[13] Yanqiao Zhu, Weizhi Xu, Jinghao Zhang, Yuanqi Du, Jieyu Zhang, Qiang Liu, Carl Yang, and Shu Wu. 2018. A Survey on Graph Structure Learning: Progress and Opportunities. *Advances in neural information processing systems* 31 (2018).