ENHANCING LINK PREDICTION ON GRAPHS: A MULTIFACETED APPROACH

By

Harry Shomer

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Computer Science—Doctor of Philosophy

2025

ABSTRACT

Graphs are a common way of representing real-world structured data. A graph is composed of nodes connected with one another via edges (i.e., "links"), where a link models how such nodes are related to one another. Due to the prevalence of graph-structured data, machine learning on graph data has become very popular.

Link prediction, which attempts to predict unseen links in a graph, is a fundamental task on graphs. Link prediction has a multitude of real-world applications, including in recommender systems, knowledge graphs, and biology. In recent years, a flurry of methods have been introduced that make use of graph neural networks (GNNs) for this task. However, we find that multiple limitations impede our ability to create effective link prediction models that can perform in real-world settings. First, we find that the current method of evaluating link prediction models is both unrealistic and too easy, resulting in inflated model performance that doesn't reflect real-world performance. Second, we observe that current methods are limited in their ability to model various patterns in link formation. This poses a challenge in real-world datasets where links can form for a number of reasons. Third, efficiency is an important concern in link prediction, as effective models are often expensive to run. Lastly, we find that link prediction models on knowledge graphs suffer from degree bias, where poor representations are learnt for lower-degree nodes, leading to subpar performance.

In this thesis, we first uncover the root cause of these fundamental limitations. I will then introduce our attempts to combat these problems, through the design of new evaluation strategies and new model design. Through the introduction of these new approaches, we help promote better use of link prediction in more realistic scenarios that can occur in the wild.

To my parents and family for their support.

ACKNOWLEDGEMENTS

During my PhD, I have been lucky to have been mentored and supported by many different people.

First, I'd like to thank my advisor Dr. Jiliang Tang, for his advice, encouragement, and support throughout my five years at MSU. In many ways, I got incredibly lucky to have stumbled into having Dr. Tang as my advisor. Before joining Dr. Tang's lab, I had no prior research experience. In fact, I knew absolutely nothing about Graph Machine Learning. Due to my lack of research experience, I had a lot of learning to do. Nonetheless, Dr. Tang was very patient with me, standing by me as I struggled to get my first paper accepted (which didn't occur until the middle of my third year). He has imparted onto me an immeasurable amount of wisdom, not only in the area of research but in mentoring and resilience. I will be forever grateful for his help and support in helping shape me into the researcher I am today.

I would like to express my gratitude to my PhD committee members: Dr. Tai-Quan "Winson" Peng, Dr. Kevin Liu, and Dr. Neil Shah. Their helpful questions, comments, and encouragement were greatly appreciated.

I have been lucky to have had many amazing and supportive colleagues during my PhD. First, I am deeply thankful to the many colleagues and labmates in the Data Science & Engineering (DSE) lab at MSU. This includes: Dr. Yao Ma, Dr. Hamid Karimi, Dr. Jamell Dacon, Dr. Wenqi Fan, Dr. Tyler Derr, Yaxin Li, Dr. Haochen Liu, Dr. Hua Liu, Dr. Xiaorui Liu, Dr. Wei Jin, Dr. Wentao Wang, Dr. Xiaoyang Wang, Dr. Yiqi Wang, Dr. Zhiwei Wang, Dr. Han Xu, Dr. Xiangyu Zhao, Dr. Hongzhi Wen, Pengfei He, Dr. Haitao Mao, Dr. Juanhui Li, Jiayuan Ding, Haoyu Han, Yingqian Cui, Jie Ren, Kaiqi Yang, Hang Li, Yuxuan Wan, Zhikai Chen, Kai Guo, Wenzhuo Tang, Dr. Remy Liu, Yuping Lin, Jay Revolinsky, Yu Song, Jingzhe Liu, Xinnan Dai, Hanbing Wang, Shenglai Zeng, Li Ma, Shen Dong, Quang Truong, Bingheng Li, Bo Wang, Yucheng Chu. In particular, I'd like to emphasize Dr. Yao Ma and Dr. Wei Jin, who both provided valuable encouragement and advice as mentors during my first couple of years at MSU. I'd also like to extend my thanks to my other collaborators, including: Dr. Geri Skenderi, Dr. Bo Wu, Dr. Jiejun Xu, Dr. Neil Shah, Dr. Yu Wang, Dr. Hui Liu, Jiawei Xu, Dr. Yasemin Copur-Gencturk, Dr. Zhigang Hua, Dr. Tong

Zhao, Dr. Amin Javari, Dr. Charu C. Aggarawl, and Dr. Shaylynn Crum-Dacon.

I would also like to give a special thank you to the members of the Shiu Lab at MSU. I spent a lot of time at the Shiu Lab and am incredibly appreciative for being considered a “honorary labmate”. In particular I’d to thank Dr. Shinhan Shiu and Dr. Melissa Lehti-Shiu for their support and for inviting me to their house for many lab parties. I’d also like to thank the various members of the lab, including: Kenia Segura Abá, Brianna Brown, Thilanka Ranaweera, Edmond Anderson, Dr. Huan Chen, and Dr. Jyothi Kumar. I am beyond grateful to consider all of you my friends.

Last, but not least, I’d like give my upmost thanks to my family and friends for their unconditional love and support.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Motivation

Graphs are ubiquitous data structures, being used to represent data across many domains. Recent work in Graph ML has shown tremendous promise in modeling graphs [55] in many real-world applications such as in the sciences and finance. One common downstream task on graphs is link prediction (LP), which attempts to predict whether two nodes in a graph are connected. LP has a wide span of real-world applications ranging from recommender systems [46], biological networks [57], and knowledge graph completion [100].

However, several key challenges prevent wider adoption of such models: **(1) Bias & Efficiency:** Graph ML models are known to often suffer from bias [103] and poor efficiency [20]. Fixing these problems, requires understanding their root causes at both the model and data level. **(2) Performance on Diverse Data:** To be viable for widespread use, graph-based models must be performant on data from a diverse set of domains [72]. This is necessary to promote widespread adoption of those models. **(3) Real-World Evaluation:** To correctly judge the effectiveness of different models, model evaluation must correspond to real-world model usage. However, this is not always the case [62], resulting in a mismatch between offline evaluation and actual usage. All these problems come together to present a set of unique challenges to using LP in realistic settings.

To help solve the varied and *multifaceted* set of challenges faced by LP, we propose to use data-centric approach. By *data-centric*, we mean that the problems are in fact related to the data itself, and can be solved through an approach that emphasizes the data. In particular, **(1)** By analyzing the data, we can glean valuable insights into the core problems that effect performance, efficiency, and bias. **(2)** These insights can then be leveraged to motivate the design of more scalabale and effective models. **(3)** Furthermore, by understanding the data, we can determine if our evaluation is aligned with actual use of these models. Through this, we can then introduce newer evaluation procedures or benchmark datasets that better reflect realistic model performance.

Next, I describe how this framework is applied to enrich our ability to perform link prediction

on graphs.

1.2 Contributions

My research applies the aforementioned framework to enhancing the task of link prediction (LP). Specifically, my work concerns advancing LP in two main areas. The first is LP on uni-relational graphs from various domains (this is typically just referred to as "link prediction" in literature, a convention we adhere to). The second is LP on knowledge graphs (KGs), which is generally referred to as KG completion (KGC). KGs are a type of multi-relational graph that encodes facts as links. As such, LP on KGs is analogous to predicting new facts. For each task, we use the knowledge gained from understanding the data to promote more effective models and evaluation procedures. Our contributions are as follows:

1. We observe that there are several issues that plague the existing evaluation of models for LP. First, there is a lack of unified settings used across papers, resulting in different evaluation metrics, data splits, and training procedures being used. This complicates any one-to-one comparison between models. Second, we find that the previous evaluation setting is unrealistic and doesn't correspond to real-world evaluation of LP. To address these issues, we first conduct a benchmarking of all prominent models and datasets under the same settings, to better compare performance. We then propose a new evaluation setting, **HeaRT**, which more closely aligns with how LP is conducted. We find that the previous evaluation procedure greatly overestimates the performance of LP methods.
2. Work has shown that multiple types of factors need to be used to adequately perform LP [72]. However, most existing methods are either unable to model all such factors. Furthermore, those that can are prohibitively expensive. In pursuit of an expressive and scalable model, we propose **LPFormer**, which efficiently and adaptively stresses different LP factors for each link being predicted (i.e., the "target link"). We achieve this by considering the relationship between both nodes in the target link in the context of the graph. The relationship is extracted via a learnable attention mechanism between the target link and the nodes in the graph. Using attention allows for the factors to vary by target link, thereby stressing different underlying factors for different

target links. We demonstrate the ability of LPFormer on both the original and HeaRT evaluations settings.

3. A common concern when learning on graphs is degree bias. This causes graph algorithms to learn poorer representations for nodes of lower degree, thereby degrading their performance on downstream tasks. This is harmful as the final learnt model will be biased against nodes of lower degree. Motivated by this problem we asked – *how does degree bias effect KG completion?* We observed that unlike other graph tasks, it is not the node degree that best correlates with performance on KGs but rather the number of links where the relation and the node being predicted co-occur together. To obviate this special type of degree bias, we propose a model-agnostic data augmentation technique, **KG-Mixup**, which creates additional synthetic links for those samples of lower degree and can empirically improve performance.
4. Recently, GNN-based models that propagate path information, "path-based GNNs", have gained prominence for KG reasoning. But, to achieve satisfying performance, they require a large number of layers, rendering them inefficient in real-world applications. However, we observe that such models tend to propagate messages that either have redundant path information or contain no information (i.e., "empty"). From these observations, we propose a new model **TAGNet**, that works by reducing the number of redundant or empty messages propagated via a set of constraints on the messages passed. We observe that TAGNet can propagate up to 90% less messages than baseline models, while achieving similar or even better performance than other models.

1.3 Outline

In Chapter 2, we observe that the current evaluation setting for LP is unrealistic. We then propose a new method – HeaRT, which more closely resembles LP in real-world application. In Chapter 3, we introduce a new method for LP, which uses a Transformer architecture to model the various underlying link-formation mechanisms. In Chapter 4, we study the problem of degree bias in KG completion. We show that it manifests itself differently from other graph tasks. We then propose a simple data augmentation strategy, KG-Mixup, to help alleviate the bias. In Chapter 5,

we aim to improve the efficiency of path-based GNNs for KG completion. We show that existing methods tend to propagate many unnecessary messages, a propose a new method TAGNet to remove them. Lastly, we conclude the dissertation in Chapter 6 with a discussion of future research directions.

CHAPTER 2

EVALUATING GRAPH NEURAL NETWORKS FOR LINK PREDICTION: CURRENT PITFALLS AND NEW BENCHMARKING

2.1 Introduction

The task of link prediction is to determine the existence of an edge between two unconnected nodes in a graph. Existing link prediction algorithms attempt to estimate the proximity of different pairs of nodes in the graph, where node pairs with a higher proximity are more likely to interact [69]. Link prediction is applied in many different domains including social networks [27], biological networks [57], and recommender systems [46].

Graph neural networks (GNNs) [125] have gained prominence in recent years with many new frameworks being proposed for a variety of different tasks. Corresponding to the rise in popularity of GNNs, there has been a number of studies that attempt to critically examine the effectiveness of different GNNs on various tasks. This can be seen for the task of node classification [101], graph classification [34], knowledge graph completion (KGC) [95, 3, 105], and others [32].

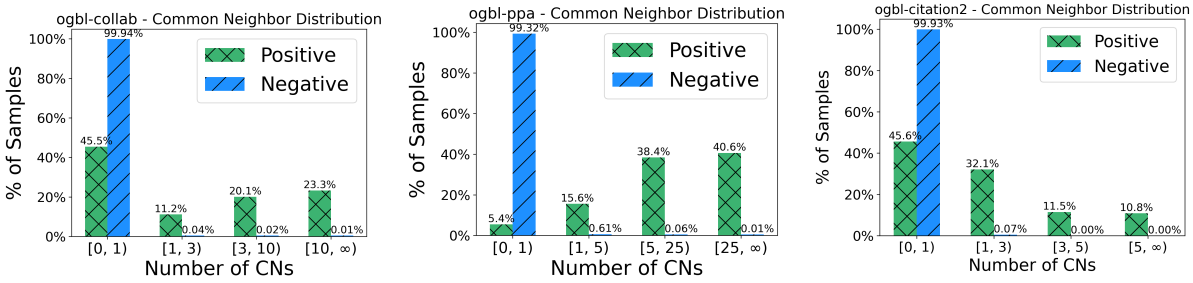
However, despite a number of new GNN-based methods being proposed [139, 20, 136, 120] for link prediction, there is currently no work that attempts to carefully examine recent advances in link prediction methods. Upon examination, we find that there are several pitfalls in regard to model evaluation that impede our ability to properly evaluate current methods. This includes:

1. **Lower than Actual Performance**. We observe that the current performance of multiple models is underreported. For some methods, such as standard GNNs, this is due to poor hyperparameter tuning. Once properly tuned, they can even achieve the best overall performance on some metrics (see SAGE [42] in Table 2.1). Furthermore, for other methods like Neo-GNN [136] we can achieve around an 8.5 point increase in Hits@50 on ogbl-collab relative to the originally reported performance. This results in Neo-GNN achieving the best overall performance on ogbl-collab in our study (see Table 2.2). Such problems obscure the true performance of different models, making it difficult to draw reliable conclusions from the current results.
2. **Lack of Unified Settings**. For Cora, Citeseer, and Pubmed datasets [131], there exists no

unified data split and evaluation metrics used for each individually. For the data split, some works [115, 153] use a single fixed train/valid/test split with percentages 85/5/10%. More recent works [20, 120] use 10 random splits of size 70/10/20%. In terms of the evaluation metrics, some studies [20, 120] use ranking-based metrics such as MRR or Hits@K while others [54, 153] report the area under the curve (AUC). This is despite multiple studies that argue that AUC is a poor metric for evaluating link prediction [129, 47]. Additionally, for both the planetoid (i.e., Cora, Citeseer and Pubmed) and ogbl-collab datasets, some methods incorporate the validation edges during testing [20, 43], while others [136, 120] don't. This lack of a unified setting makes it difficult to draw a comparison and hampers our ability to determine which methods perform best on these datasets.

3. **Unrealistic Evaluation Setting**. During the evaluation, we are given a set of true samples (i.e., positive samples) and a set of false samples (i.e., negative samples). We are tasked with learning a classifier f that assigns a higher probability to the positive samples than the negatives. The current evaluation setting uses the same set of randomly selected negative samples for each positive sample. We identify two potential problems with the current evaluation procedure. **(1)** It is not aligned with real-world settings. In a real-world scenario, we typically care about predicting links for a specific node. For example, in friend recommendations, we aim to recommend friends for a specific user u. To evaluate such models for u, we strive to rank node pairs including u. However, this does not hold in the current setting as u is not included in most of the negative samples. **(2)** The current evaluation setting makes the task too easy. As such, it may not reflect the model performance in real-world applications. This is because the nodes in a randomly selected negative "node pair" are likely to be unrelated to each other. As shown in Figure 2.1, almost all negative samples in the test data have no common neighbors, a typically strong heuristic, making them trivial to classify them.

To account for these issues, we propose to first conduct a fair and reproducible evaluation among current link prediction methods under the existing evaluation setting. We then design a new evaluation strategy that is more aligned with a real-world setting and detail our results. Our key



(a) ogbl-collab (b) ogbl-ppa (c) ogbl-citation2

Figure 2.1 Common neighbor distribution for the positive and negative test samples for the ogbl-collab, ogbl-ppa, and ogbl-citation2 datasets under the existing evaluation setting.

contributions are summarized below:

1. **Reproducible and Fair Comparison**. We conduct a fair comparison of different models across multiple common datasets. To ensure a fair comparison, we tune all models on the same set of hyperparameters. We further evaluate different models using multiple types of evaluation metrics. For the Planetoid datasets [131], we further use a unified data split to facilitate a point of comparison between models. To the best of our knowledge, there are no recent efforts to comprehensively benchmark link prediction methods (several exist for KGC [105, 3, 95]). Furthermore, we open-source the implementation in our analysis to enable others in their analyses.
2. **New Evaluation Setting**. We recognize that the current negative sampling strategy used in evaluation is unrealistic and easy. To counter these issues, we first use a more realistic setting of tailoring the negatives to each positive sample. This is achieved by restricting them to be corruptions of the positive sample (i.e., containing one of its two nodes as defined in Eq. (2.3)). Given the prohibitive cost of utilizing all possible corruptions, we opt instead to only rank against K negatives for each positive sample. In order to choose the most relevant and difficult corruptions, we propose a **He**uristic **R**elated Sampling **T**echnique (HeaRT), which selects them based on a combination of multiple heuristics. This creates a more challenging task than the previous evaluation strategy and allows us to better assess the capabilities of current methods.

The rest of the paper is structured as follows. In Section 2.2 we introduce the models, datasets, and settings used for conducting a fair comparison between methods. In Section 2.3 we show the results of the fair comparison under the existing evaluation setting and discuss our main observations. Lastly, in Section 2.4 we introduce our new evaluation setting. We then detail and discuss the performance of different methods using our new setting.

2.2 Preliminaries

2.2.1 Task Formulation

In this section we formally define the task of link prediction. Let $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$ be a graph where $\mathcal{V}$ and $\mathcal{E}$ are the set of nodes and edges in the graph, respectively. Furthermore, let $X \in \mathcal{R}^{|V| \times d}$ be a set of d-dimensional features for each node. Link prediction aims to predict the likelihood of a link existing between two nodes given the structural and feature information. For a pair of nodes u and v, the probability of a link existing, $p(u, v)$, is therefore given by:

$$p(u, v) = p(u, v \mid \mathcal{G}, X). \tag{2.1}$$

Traditionally, $p(u, v)$ was estimated via non-learnable heuristic methods [74, 65]. More recently, methods that use learnable parameters have gained popularity [139, 20]. These methods attempt to estimate $p(u, v)$ via a learnable function f such that:

$$p(u, v) = f(u, v \mid \mathcal{G}, X, \Theta), \tag{2.2}$$

where Θ represents a set of learnable parameters. A common choice of f are graph neural networks [55]. In the next subsection we detail the various link prediction methods used in this study.

2.2.2 Link Prediction Methods

In this section we given an overview of the different methods used in this study. Conventional methods [74, 65] often exploit hand-craft graph structural properties (i.e., heuristics) between node pairs. GNNs attempt to learn the structural information to facilitate link prediction [141, 120, 20]. Given the strong performance of pairwise-based heuristics [136, 120], some recent works use both GNNs and pairwise information, demonstrating strong performance.

For our study, we consider both traditional and state-of-the-art GNN-based models. They can be roughly organized into four categories. **1) Heuristic methods**: Common Neighbor (CN) [81], Adamic Adar (AA) [2], Resource Allocation (RA) [151], Shortest Path [65], and Katz [49]. These methods define a score to indicate the link existence based on the graph structure. Among them, CN, AA, and RA are based on the common neighbors, while Shortest Path and Katz are based on the path information. **2) Embedding methods**: Matrix factorization (MF) [74], Multilayer Perceptron (MLP) and Node2Vec [39]. These methods are trained to learn low-dimensional node embeddings that are used to predict the likelihood of node pairs existing. **3) GNN methods**: GCN [132], GAT [115], SAGE [42], and GAE [54]. These methods attempt to integrate the multi-hop graph structure based on the message passing paradigm. **4) GNN + Pairwise Information methods**: Standard GNN methods, while powerful, are not able to capture link-specific information [141]. As such, works have been proposed that augment GNN methods by including additional information to better capture the relation between the nodes in the link we are predicting. SEAL [141], BUDDY [20], and NBFNet [153] use the subgraph features. Neo-GNN [136], NCN [120], and NCNC [120] are based on common neighbor information. Lastly, PEG [117] uses the positional encoding derived from the graph structure.

2.2.3 Datasets and Experimental Settings

In this section we summarize the datasets and evaluation and training settings. We note that the settings depend on the specific dataset. More details are given in Appendix A.3.

Datasets. We limit our experiments to homogeneous graphs, which are the most commonly used datasets for link prediction. This includes the small-scale datasets, i.e., Cora, Citeseer, Pubmed [131], and large-scale datasets in the OGB benchmark [43], i.e., ogbl-collab, ogbl-ddi, ogbl-ppa, and ogbl-citation2. We summarize the statistics and split ratio of each dataset in Appendix A.3.

Metrics. For evaluation, we use both the area under the curve (AUC) and ranking-based metrics, i.e., mean reciprocal rank (MRR) and Hits@K. For Cora, Citeseer, and Pubmed we adopt $K \in \{1, 3, 10, 100\}$. We note that $K = 100$ is reported in some recent works [20, 120]). However due to the small number of negatives used during evaluation (e.g., $\approx$ 500 for Cora and Citeseer)

$K = 100$ is likely not informative. For the OGB datasets, we adopt $K \in \{20, 50, 100\}$ to keep consistent with the original study [43]. Please see Appendix A.2.1 for the formal definitions of the various evaluation metrics.

Hyperparameter Ranges. We conduct a grid hyperparameter search across a comprehensive range of values. For Cora, Citeseer, and Pubmed this includes: learning rate (0.01, 0.001), dropout (0.1, 0.3, 0.5), weight decay (1e-4, 1e-7, 0), number of model layers (1, 2, 3), number of prediction layers (1, 2, 3), and the embedding size (128, 256). Due to the large size of the OGB datasets, it's infeasible to tune over such a large range. Therefore, following the most commonly used settings among published hyperparameters, we fix the weight decay to 0, the number of model and prediction layers to be 3, and the embedding size to be 256. The best hyperparameters are chosen based on the validation performance. We note that several exceptions exist to these ranges when they result in significant performance degradations (see Appendix A.3 for more details). We further follow the existing setting and only sample one negative sample per positive sample during training.

Existing Evaluation Settings. In the evaluation stage, the same set of randomly sampled negatives are used for all positive samples. We note that one exception is ogbl-citation2, where they randomly sample 1000 negative samples per positive sample. For Cora, Citeseer, and Pubmed the number of negative samples is equal to the number of positive samples. For the OGB datasets, we use the existing fixed set of randomly chosen negatives found in [43]. Furthermore, for ogbl-collab we follow the existing protocol [43] and include the validation edges in the training graph during testing. This setting is adopted on ogbl-collab under both the existing and new evaluation setting.

2.3 Fair Comparison Under the Existing Setting

In this section, we conduct a fair comparison among link prediction methods. This comparison is spurred by the multiple pitfalls noted in Section 5.1, which include lower-than-actual model performance, multiple data splits, and inconsistent evaluation metrics. These pitfalls hinder our ability to fairly compare different methods. To rectify this, we conduct a fair comparison adhering to the settings listed in section 2.2.3.

Table 2.1 Results on Cora, Citeseer, and Pubmed(%) under the existing evaluation setting. We highlight the results ranked first, second, and third as green, blue, and orange, respectively.

	Models	Cora		Citeseer		Pubmed	
		MRR	AUC	MRR	AUC	MRR	AUC
Heuristic	CN	20.99	70.85	28.34	67.49	14.02	63.9
	AA	31.87	70.96	29.37	67.49	16.66	63.9
	RA	30.79	70.96	27.61	67.48	15.63	63.9
	Shortest Path	12.45	81.08	31.82	75.5	7.15	74.64
	Katz	27.4	81.17	38.16	75.37	21.44	74.86
Embedding	Node2Vec	**37.29 ± 8.82**	90.97 ± 0.64	44.33 ± 8.99	94.46 ± 0.59	34.61 ± 2.48	93.14 ± 0.18
	MF	14.29 ± 5.79	80.29 ± 2.26	24.80 ± 4.71	75.92 ± 3.25	19.29 ± 6.29	93.06 ± 0.43
	MLP	31.21 ± 7.90	95.32 ± 0.37	43.53 ± 7.26	94.45 ± 0.32	16.52 ± 4.14	98.34 ± 0.10
GNN	GCN	32.50 ± 6.87	95.01 ± 0.32	50.01 ± 6.04	95.89 ± 0.26	19.94 ± 4.24	98.69 ± 0.06
	GAT	31.86 ± 6.08	93.90 ± 0.32	48.69 ± 7.53	96.25 ± 0.20	18.63 ± 7.75	98.20 ± 0.07
	SAGE	**37.83 ± 7.75**	**95.63 ± 0.27**	47.84 ± 6.39	**97.39 ± 0.15**	22.74 ± 5.47	**98.87 ± 0.04**
	GAE	29.98 ± 3.21	95.08 ± 0.33	**63.33 ± 3.14**	**97.06 ± 0.22**	16.67 ± 0.19	97.47 ± 0.08
GNN+Pairwise Info	SEAL	26.69 ± 5.89	90.59 ± 0.75	39.36 ± 4.99	88.52 ± 1.40	**38.06 ± 5.18**	97.77 ± 0.40
	BUDDY	26.40 ± 4.40	95.06 ± 0.36	**59.48 ± 8.96**	96.72 ± 0.26	23.98 ± 5.11	98.2 ± 0.05
	Neo-GNN	22.65 ± 2.60	93.73 ± 0.36	53.97 ± 5.88	94.89 ± 0.60	31.45 ± 3.17	98.71 ± 0.05
	NCN	32.93 ± 3.80	**96.76 ± 0.18**	54.97 ± 6.03	97.04 ± 0.26	**35.65 ± 4.60**	**98.98 ± 0.04**
	NCNC	29.01 ± 3.83	**96.90 ± 0.28**	**64.03 ± 3.67**	**97.65 ± 0.30**	25.70 ± 4.48	**99.14 ± 0.03**
	NBFNet	**37.69 ± 3.97**	92.85 ± 0.17	38.17 ± 3.06	91.06 ± 0.15	**44.73 ± 2.12**	98.34 ± 0.02
	PEG	22.76 ± 1.84	94.46 ± 0.34	56.12 ± 6.62	96.15 ± 0.41	21.05 ± 2.85	96.97 ± 0.39

The results are split into two tables. The results for Cora, Citeseer, and Pubmed are shown in Table 2.1 and OGB in Table 2.2. For simplicity, we only present the AUC and MRR for Cora, Citeseer, and Pubmed. For OGB datasets, we include AUC and the original ranking metric reported in [43] to allow a convenient comparison (Hits@20 for ogbl-ddi, Hits@50 for ogbl-collab, Hits@100 for ogbl-ppa, and MRR for ogbl-citation2). We use ">24h" to denote methods that require more than 24 hours for either training one epoch or evaluation. OOM indicates that the algorithm requires over 50Gb of GPU memory. Since ogbl-ddi has no node features, we mark the MLP results with a "N/A". Additional results in terms of other metrics are presented in Appendix A.6. We have several noteworthy observations concerning the methods, the datasets, the evaluation settings, and the overall results. We highlight the main observations below.

Observation 1: Better than Reported Performance. We find that for some models we are able to achieve superior performance compared to what is reported by recent studies. For instance, in our study Neo-GNN [136] achieves the best overall test performance on ogbl-collab with a Hits@50 of 66.13. In contrast, the reported performance in [136] is only 57.52, which would rank seventh under our current setting. This is because the original study [136] does not follow the standard setting

Table 2.2 Results on OGB datasets (%) under the existing evaluation setting. We highlight the results ranked first, second, and third as green, blue, and orange, respectively.

	Models	ogbl-collab		ogbl-ddi		ogbl-ppa		ogbl-citation2
		Hits@50	AUC	Hits@20	AUC	Hits@100	AUC	MRR
Heuristic	CN	61.37	82.78	17.73	95.2	27.65	97.22	74.3
	AA	64.17	82.78	18.61	95.43	32.45	97.23	75.96
	RA	63.81	82.78	6.23	96.51	**49.33**	97.24	76.04
	Shortest Path	46.49	96.51	0	59.07	0	99.13	>24h
	Katz	64.33	90.54	17.73	95.2	27.65	97.22	74.3
Embedding	Node2Vec	49.06 ± 1.04	96.24 ± 0.15	34.69 ± 2.90	99.78 ± 0.04	26.24 ± 0.96	99.77 ± 0.00	45.04 ± 0.10
	MF	41.81 ± 1.67	83.75 ± 1.77	23.50 ± 5.35	99.46 ± 0.10	28.4 ± 4.62	99.46 ± 0.10	50.57 ± 12.14
	MLP	35.81 ± 1.08	95.91 ± 0.08	N/A	N/A	0.45 ± 0.04	90.23 ± 0.00	38.07 ± 0.09
GNN	GCN	54.96 ± 3.18	97.89 ± 0.06	**49.90 ± 7.23**	99.86 ± 0.03	29.57 ± 2.90	**99.84 ± 0.03**	84.85 ± 0.07
	GAT	55.00 ± 3.28	97.11 ± 0.09	31.88 ± 8.83	99.63 ± 0.21	OOM	OOM	OOM
	SAGE	59.44 ± 1.37	**98.08 ± 0.03**	49.84 ± 15.56	**99.96 ± 0.00**	41.02 ± 1.94	99.82 ± 0.00	83.06 ± 0.09
	GAE	OOM	OOM	7.09 ± 6.02	75.34 ±15.96	OOM	OOM	OOM
GNN+Pairwise Info	SEAL	63.37 ± 0.69	95.65 ± 0.29	25.25 ± 3.90	97.97 ± 0.19	48.80 ± 5.61	99.79 ± 0.02	86.93 ± 0.43
	BUDDY	**64.59 ± 0.46**	96.52 ± 0.40	29.60 ± 4.75	99.81 ± 0.02	47.33 ± 1.96	99.56 ± 0.02	**87.86 ± 0.18**
	Neo-GNN	**66.13 ± 0.61**	**98.23 ± 0.05**	20.95 ± 6.03	98.06 ± 2.00	48.45 ± 1.01	97.30 ± 0.14	83.54 ± 0.32
	NCN	63.86 ± 0.51	97.83 ± 0.04	**76.52 ± 10.47**	**99.97 ± 0.00**	**62.63 ± 1.15**	**99.95 ± 0.01**	**89.27 ± 0.05**
	NCNC	**65.97 ± 1.03**	**98.20 ± 0.05**	**70.23 ± 12.11**	**99.97 ± 0.01**	**62.61 ± 0.76**	**99.97 ± 0.01**	**89.82 ± 0.43**
	NBFNet	OOM	OOM	>24h	>24h	OOM	OOM	OOM
	PEG	49.02 ± 2.99	94.45 ± 0.89	30.28 ± 4.92	99.45 ± 0.04	OOM	OOM	OOM

of including validation edges in the graph during testing. This setting, as noted in Section 2.2.3, is used by all other methods on ogbl-collab. However it was omitted by [136], resulting in lower reported performance. Furthermore, on ogbl-citation2 [43], our results for the heuristic methods are typically around 75% MRR. This significantly outperforms previously reported results, which report an MRR of around 50% [141, 20]. The disparity arises as previous studies treat the ogbl-citation2 as a directed graph when applying heuristic methods. However, for GNN-based methods, ogbl-citation2 is typically converted to a undirected graph. We remedy this by also converting ogbl-citation2 to an undirected graph when computing the heuristics, leading to a large increase in performance.

Furthermore, with proper tuning, conventional baselines like GCN [55] and GAE [54] generally exhibit enhanced performance relative to what was originally reported across all datasets. For example, we find that GAE can achieve the second best MRR on Citeseer and GCN the third best Hits@20 on ogbl-ddi. A comparison of the reported results and ours are shown in Table 2.3. We note that we report AUC for Cora, Citeseer, Pubmed as it was used in the original study. These observations suggest that the performance of various methods are better than what was reported in their initial publications. However, many studies [20, 120, 141] only report the original performance

Table 2.3 Comparison of ours and the reported results for GCN and GAE.

GCN	ogbl-collab Hits@50	ogbl-ppa Hits@100	ogbl-ddi Hits@20	ogbl-citation2 MRR	GAE	Cora AUC	Citeseer AUC	Pubmed AUC
Reported	47.14 ± 1.45	18.67 ± 1.32	37.07 ± 5.07	84.74 ± 0.21	Reported	91.00 ± 0.01	89.5 ± 0.05	96.4 ± 0.00
Ours	**54.96 ± 3.18**	**29.57 ± 2.90**	**49.90 ± 7.23**	**84.85 ± 0.07**	Ours	**95.08 ± 0.33**	**97.06 ± 0.22**	**97.47 ± 0.08**

for comparison, which has the potential to lead to inaccurate conclusions.

Observation 2: Divergence from Reported Results on ogbl-ddi. We observe that our results in Table 2.2 for ogbl-ddi differ from the reported results. Outside of GCN, which reports better performance, most other GNN-based methods report a lower-than-reported performance. For example, for BUDDY we only achieve a Hits@20 of 29.60 vs. the reported 78.51 (see Appendix A.4 for a comprehensive comparison among methods). We find that the reason for this difference depends on the method. BUDDY [20] reported [1] using 6 negatives per positive sample during training, leading to an increase in performance. Neo-GNN [136] first pretrains the GNN under the link prediction task, and then uses the pretrained model as the initialization for Neo-GNN.[2] For a fair comparison among methods, we only use 1 negative per positive sample in training and we don't apply the pretraining. For other methods, we find that a weak relationship between the validation and test performance complicates the tuning process, making it difficult to find the optimal hyperparameters. Please see Appendix A.5 for a more in-depth study and discussion.

Observation 3: High Model Standard Deviation. The results in Tables 2.1 and 2.2 present the mean performance and standard deviation when training over 10 seeds. Generally, we find that for multiple datasets the standard deviation of the ranking metrics is often high for most models. For example, the standard deviation for MRR can be as high as 8.82, 8.96, or 7.75 for Cora, Citeseer, and Pubmed, respectively. Furthermore, on ogbl-ddi the standard deviation of Hits@20 reaches as high as 10.47 and 15.56. A high variance indicates unstable model performance. This makes it difficult to compare results between methods as the true performance lies in a larger range. This further complicates replicating model performance, as even large differences with the reported results may still fall within variance (see observation 2). Later in Section 2.4.3 we find that our

[1] https://github.com/melifluos/subgraph-sketching

[2] https://github.com/seongjunyun/Neo-GNNs

new evaluation can reduce the model variance for all datasets (see Table 2.6). This suggests that the high variance is related to the current evaluation procedure.

Observation 4: Inconsistency of AUC vs. Ranking-Based Metrics. The AUC score is widely adopted to evaluate recent advanced link prediction methods [54, 153]. However, from our results in Tables 2.1 and 2.2 we observe that there exists a disparity between AUC and ranking-based metrics. In some cases, the AUC score can be high when the ranking metric is very low or even 0. For example, the Shortest Path heuristic records a Hits@K of 0 on ogbl-ppa. However, the AUC score is $> 99\%$. Furthermore, even though RA records the third and fifth best performance on ogbl-ppa and ogbl-collab, respectively, it has a lower AUC score than Shortest Path on both. Previous works [47, 129] argued that AUC is not a proper metric for link prediction. This is due to the inapplicability of AUC for highly imbalanced problems [28, 99].

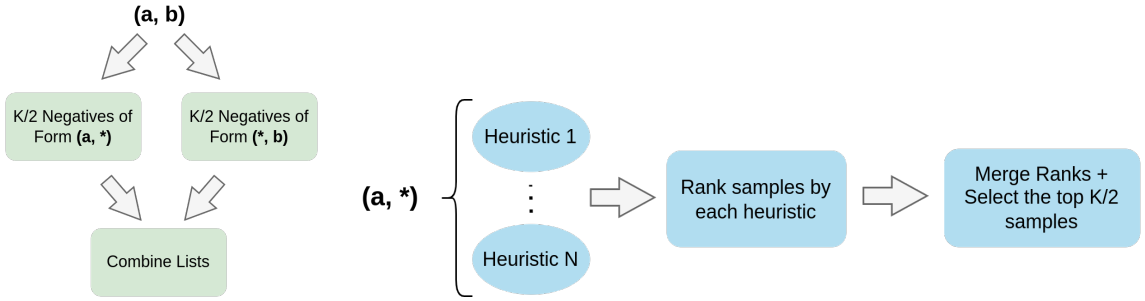
2.4 New Evaluation Setting

In this section, we introduce a new setting for evaluating link prediction methods. We first discuss the unrealistic nature of the current evaluation setting in Section 2.4.1. We then present our new evaluation setting in Section 2.4.2, which aims to align better with real-world scenarios. Lastly, in Section 2.4.3, we present and discuss the results based on our new evaluation setting.

2.4.1 Issues with the Existing Evaluation Setting

The existing evaluation procedure for link prediction is to rank a positive sample against a set of K randomly selected negative samples. The same set of K negatives are used for all positive samples (with the exception of ogbl-citation2 which uses 1000 per positive sample). We demonstrate that there are multiple issues with this setting, making it difficult to properly evaluate the effectiveness of current models.

Issue 1: Non-Personalized Negative Samples. The existing evaluation setting uses the same set of negative samples for all positive samples (outside of ogbl-citation2). This strategy, referred to as global negative sampling [123], is not a commonly sought objective. Rather, we are often more interested in predicting links that will occur for a specific node. Take, for example, a social network that connects users who are friends. In this scenario, we may be interested in recommending new



(a) Negative sample generation for one positive sample.

(b) Process of determining negative samples that contain a node a.

Figure 2.2 Pipeline for generating the hard negative samples for a positive sample (a, b).

friends to a user u. This requires learning a classifier f that assigns a probability to a link existing. When evaluating this task, we want to rank links where u connects to an existing friend above those where they don't. For example, if u is friends with a but not b, we hope that $f(u, a) > f(u, b)$. However, the existing evaluation setting doesn't explicitly test for this. Rather it compares a true sample (u, a) with a potentially unrelated negative sample, e.g., (c, d). This is not aligned with the real-world usage of link prediction on such graphs.

Issue 2: Easy Negative Samples. The existing evaluation setting randomly selects negative samples to use. However given the large size of most graphs (see Table A.1 in Appendix A.3), randomly sampled negatives are likely to choose two nodes that bear no relationship to each other. Such node pairs are trivial to classify. We demonstrate this by plotting the distribution of common neighbors (CN), a strong heuristic, for all positive and negative test samples in Figure 2.1. Almost all the negative samples contain no CNs, making them easy to classify. We further show that the same problem afflicts even the smaller datasets in Figure A.1 in Appendix A.1.

These observations suggest that a more realistic evaluation strategy is desired. At the core of this challenge is which negative samples to use during evaluation. We discuss our design for solving this in the next subsection.

2.4.2 Heuristic Related Sampling Technique (HeaRT)

In this subsection, we introduce new strategy for evaluating link prediction methods. To address the concerns outlined in Section 2.4.1, we design a new method for sampling negatives during evaluation. Our strategy, HeaRT, solves these challenges by: (a) personalizing the negatives to each sample and (b) using heuristics to select hard negative samples. This allows for the negative samples to be directly related to each positive sample while also being non-trivial. We further discuss how to ensure that the negative samples are both *personalized* and *non-trivial* for a specific positive sample.

From our discussion in Section 2.4.1, we are motivated in personalizing the negatives to each positive sample. Since the positive samples in the current datasets are node pairs, we seek to personalize the negatives to both nodes in the positive sample. Extending our example in Section 2.4.1, this is analogous to restricting the negatives to contain one of the two users from the original friendship pair. As such, for a positive sample (u, a), the negative samples will belong to the set:

$$S(u, a) = \{(u', a) \mid u' \in \mathcal{V}\} \cup \{(u, a') \mid a' \in \mathcal{V}\}, \tag{2.3}$$

where $\mathcal{V}$ is the set of nodes. This is similar to the setting used for knowledge graph completion (KGC) [13] which uses all such samples for evaluation. However, one drawback of evaluating each positive sample against the entire set of possible corruptions is the high computational cost. To mitigate this issue we consider only utilizing a small subset of $S(u, a)$ during evaluation.

The key challenge is how to generate a subset of $S(u, a)$. If we randomly sample from $S(u, a)$, we risk only utilizing easy negative samples. This is one of the issues of the existing evaluation setting (see Issue 2 in Section 2.4.1), whereby randomly selecting negatives, they unknowingly produce negative samples that are too easy. We address this by selecting the negative samples via a combination of multiple heuristics. Since heuristics typically correlate well with performance, we ensure that the negative samples will be non-trivial to classify. This is similar to the concept of candidate generation [41, 33], which only ranks a subset of candidates that are most likely to be true.

Table 2.4 Results on Cora, Citeseer, and Pubmed (%) under HeaRT. We highlight the results ranked first, second, and third as green, blue, and orange, respectively.

	Models	Cora		Citeseer		Pubmed	
		MRR	Hits@10	MRR	Hits@10	MRR	Hits@10
Heuristic	CN	9.78	20.11	8.42	18.68	2.28	4.78
	AA	11.91	24.10	10.82	22.20	2.63	5.51
	RA	11.81	24.48	10.84	22.86	2.47	4.9
	Shortest Path	5.04	15.37	5.83	16.26	0.86	0.38
	Katz	11.41	22.77	11.19	24.84	3.01	5.98
Embedding	Node2Vec	14.47 ± 0.60	32.77 ± 1.29	21.17 ± 1.01	45.82 ± 2.01	3.94 ± 0.24	8.51 ± 0.77
	MF	6.20 ± 1.42	15.26 ± 3.39	7.80 ± 0.79	16.72 ± 1.99	4.46 ± 0.32	9.42 ± 0.87
	MLP	13.52 ± 0.65	31.01 ± 1.71	22.62 ± 0.55	48.02 ± 1.79	6.41 ± 0.25	15.04 ± 0.67
GNN	GCN	**16.61 ± 0.30**	**36.26 ± 1.14**	21.09 ± 0.88	47.23 ± 1.88	7.13 ± 0.27	15.22 ± 0.57
	GAT	13.84 ± 0.68	32.89 ± 1.27	19.58 ± 0.84	45.30 ± 1.3	4.95 ± 0.14	9.99 ± 0.64
	SAGE	14.74 ± 0.69	34.65 ± 1.47	21.09 ± 1.15	48.75 ± 1.85	**9.40 ± 0.70**	**20.54 ± 1.40**
	GAE	**18.32 ± 0.41**	**37.95 ± 1.24**	**25.25 ± 0.82**	**49.65 ± 1.48**	5.27 ± 0.25	10.50 ± 0.46
GNN+Pairwise Info	SEAL	10.67 ± 3.46	24.27 ± 6.74	13.16 ± 1.66	27.37 ± 3.20	5.88 ± 0.53	12.47 ± 1.23
	BUDDY	13.71 ± 0.59	30.40 ± 1.18	22.84 ± 0.36	48.35 ± 1.18	7.56 ± 0.18	16.78 ± 0.53
	Neo-GNN	13.95 ± 0.39	31.27 ± 0.72	17.34 ± 0.84	41.74 ± 1.18	**7.74 ± 0.30**	**17.88 ± 0.71**
	NCN	14.66 ± 0.95	35.14 ± 1.04	**28.65 ± 1.21**	**53.41 ± 1.46**	5.84 ± 0.22	13.22 ± 0.56
	NCNC	14.98 ± 1.00	**36.70 ± 1.57**	**24.10 ± 0.65**	**53.72 ± 0.97**	**8.58 ± 0.59**	**18.81 ± 1.16**
	NBFNet	13.56 ± 0.58	31.12 ± 0.75	14.29 ± 0.80	31.39 ± 1.34	>24h	>24h
	PEG	**15.73 ± 0.39**	36.03 ± 0.75	21.01 ± 0.77	45.56 ± 1.38	4.4 ± 0.41	8.70 ± 1.26

An overview of the generation process is given in Figure 2.2. For each positive sample, we generate K negative samples. To allow personalization to both nodes in the positive sample equally, we sample $K/2$ negatives with each node. For the heuristics, we consider RA [151], PPR [15], and feature similarity. A more detailed discussion on the negative sample generation is given in Appendix A.7. It's important to note that our work centers specifically on negative sampling during the evaluation stage (validation and test). This is distinct from prior work that concerns the negatives sampled used during the training phase [130, 91]. As such, the training process remains unaffected under both the existing and HeaRT setting.

2.4.3 Results and Discussion

In this subsection we present our results when utilizing HeaRT. We follow the parameter ranges introduced in Section 2.2.3. For all datasets we use $K = 500$ negative samples per positive sample during evaluation. Furthermore for ogbl-ppa we only use a small subset of the validation and test positive samples (100K each) for evaluation. This is because the large size of the validation and test sets (see Table A.1 in Appendix A.3) makes HeaRT prohibitively expensive.

Table 2.5 Results on OGB datasets (%) under HeaRT. We highlight the results ranked first, second, and third as green, blue, and orange, respectively.

Models	ogbl-collab		ogbl-ddi		ogbl-ppa		ogbl-citation2	
	MRR	Hits@20	MRR	Hits@20	MRR	Hits@20	MRR	Hits@20
CN	4.20	16.46	6.71	38.69	25.70	68.25	17.11	41.73
AA	5.07	19.59	6.97	39.75	26.85	70.22	17.83	43.12
RA	**6.29**	**24.29**	8.70	44.01	28.34	71.50	17.79	43.34
Shortest Path	2.66	15.98	0	0	0.54	1.31	>24h	>24h
Katz	**6.31**	**24.34**	6.71	38.69	25.70	68.25	14.10	35.55
Node2Vec	4.68 ± 0.08	16.84 ± 0.17	11.14 ± 0.95	63.63 ± 2.05	18.33 ± 0.10	53.42 ± 0.11	14.67 ± 0.18	42.68 ± 0.20
MF	4.89 ± 0.25	18.86 ± 0.40	**13.99 ± 0.47**	59.50 ± 1.68	22.47 ± 1.53	70.71 ± 4.82	8.72 ± 2.60	29.64 ± 7.30
MLP	5.37 ± 0.14	16.15 ± 0.27	N/A	N/A	0.98 ± 0.00	1.47 ± 0.00	16.32 ± 0.07	43.15 ± 0.10
GCN	6.09 ± 0.38	22.48 ± 0.81	**13.46 ± 0.34**	64.76 ± 1.45	26.94 ± 0.48	68.38 ± 0.73	19.98 ± 0.35	**51.72 ± 0.46**
GAT	4.18 ± 0.33	18.30 ± 1.42	**12.92 ± 0.39**	**66.83 ± 2.23**	OOM	OOM	OOM	OOM
SAGE	5.53 ± 0.5	21.26 ± 1.32	12.60 ± 0.72	**67.19 ± 1.18**	27.27 ± 0.30	69.49 ± 0.43	**22.05 ± 0.12**	**53.13 ± 0.15**
GAE	OOM	OOM	3.49 ± 1.73	17.81 ± 9.80	OOM	OOM	OOM	OOM
SEAL	**6.43 ± 0.32**	21.57 ± 0.38	9.99 ± 0.90	49.74 ± 2.39	**29.71 ± 0.71**	**76.77 ± 0.94**	**20.60 ± 1.28**	48.62 ± 1.93
BUDDY	5.67 ± 0.36	**23.35 ± 0.73**	12.43 ± 0.50	58.71 ± 1.63	27.70 ± 0.33	71.50 ± 0.68	19.17 ± 0.20	47.81 ± 0.37
Neo-GNN	5.23 ± 0.9	21.03 ± 3.39	10.86 ± 2.16	51.94 ± 10.33	21.68 ± 1.14	64.81 ± 2.26	16.12 ± 0.25	43.17 ± 0.53
NCN	5.09 ± 0.38	20.84 ± 1.31	12.86 ± 0.78	**65.82 ± 2.66**	**35.06 ± 0.26**	**81.89 ± 0.31**	**23.35 ± 0.28**	**53.76 ± 0.20**
NCNC	4.73 ± 0.86	20.49 ± 3.97	>24h	>24h	**33.52 ± 0.26**	**82.24 ± 0.40**	19.61 ± 0.54	51.69 ± 1.48
NBFNet	OOM	OOM	>24h	>24h	OOM	OOM	OOM	OOM
PEG	4.83 ± 0.21	18.29 ± 1.06	12.05 ± 1.14	50.12 ± 6.55	OOM	OOM	OOM	OOM

The results are shown in Table 2.4 (Cora, Citeseer, Pubmed) and Table 2.5 (OGB). For simplicity, we only include the MRR and Hits@10 for Cora, Citeseer, Pubmed, and the MRR and Hits@20 for OGB. Additional results for other metrics can be found in Appendix A.8. We note that most datasets, outside of ogbl-ppa, exhibit much lower performance than under the existing setting. This is though we typically use much fewer negative samples in the new setting, implying that the negative samples produced by HeaRT are much harder. We highlight the main observations below.

Observation 1: Better Performance of Simple Models. We find that under HeaRT, "simple" baseline models (i.e., heuristic, embedding, and GNN methods) show a greater propensity to outperform their counterparts via ranking metrics than under the existing setting. Specifically, we focus on MRR in Table 2.1, 2.4, and 2.5, and the corresponding ranking-based metrics in Table 2.2. Under the existing setting, such methods only rank in the top three for any dataset a total of **5** times. However, under HeaRT this occurs **10** times. Furthermore, under the existing setting only **1** "simple" method ranks best overall while under HeaRT there are **3**. This suggests that recent advanced methods may have benefited from the easier negative samples in the existing setting.

Another interesting observation is that on ogbl-collab, heuristic methods achieve strong per-

Table 2.6 Mean model standard deviation for the existing setting and HeaRT. We use Hits@20 for ogbl-ddi, Hits@50 for ogbl-collab, Hits@100 for ogbl-ppa, and MRR otherwise.

Dataset	Existing	HeaRT	% Change
Cora	5.19	0.79	-85%
Citeseer	5.94	0.88	-85%
Pubmed	4.14	0.35	-92%
ogbl-collab	1.49	0.96	-36%
ogbl-ppa	2.13	0.36	-83%
ogbl-ddi	6.77	3.49	-48%
ogbl-citation2	1.39	0.59	-58%

formance and are able to outperform more complicated models. Specifically, we find that Katz is ranked the second and RA the third. Note that this underscores the significance of the common neighbors information (i.e., paths of length 2), as this information is incorporated in both RA and Katz. Of note is that ogbl-collab is a dynamic collaboration graph, which is different from other datasets. Because of this, the negative sampling strategy also differs slightly from the other datasets. Please see Appendix A.9 for more discussion.

Observation 2: Lower Model Standard Deviation. We observed earlier that, under the existing evaluation setting, the model variance across seeds was high (see observation 3 in Section 2.3). This complicates model comparison as the model performance is unreliable. Interestingly, we find that HeaRT is able to dramatically reduce the variance for all datasets. We demonstrate this by first calculating the mean standard deviation across all models on each individual dataset. This was done for both evaluation settings with the results compared. As demonstrated in Table 2.6, the mean standard deviation decreases for all datasets. This is especially true for Cora, Citeseer, and Pubmed, which each decrease by over 85%. Such a large decrease in standard deviation is noteworthy as it allows for a more trustworthy and reliable comparison between methods.

We posit that this observation is caused by a stronger alignment between the positive and negative samples under our new evaluation setting. Under the existing evaluation setting, the same set of negative samples is used for all positive samples. One consequence of this is that a single positive sample may bear little to no relationship to the negative samples (see Section 2.4.1 for more discussion). However, under our new evaluation setting, the negatives for a positive sample

are a subset of the corruptions of that sample. This allows for a more natural comparison via ranking-based metrics as the samples are more related and can be more easily compared.

Observation 3: Lower Model Performance. We observe that the majority of datasets exhibit a significantly reduced performance in comparison to the existing setting. For example, under the existing setting, models typically achieve a MRR of around 30, 50, and 30 on Cora, Citeseer, and Pubmed (Table 2.1), respectively. However, under HeaRT the MRR for those datasets is typically around 20, 25, and 10 (Table 2.4). Furthermore for ogbl-citation2, the MRR of the best performing model falls from a shade under 90 on the existing setting to slightly over 20 on HeaRT. Lastly, we note that the performance on ogbl-ppa actually increases. This is because we only utilize a small subset of the total test set when evaluating on HeaRT, nullifying any comparison between the two settings.

These outcomes are observed despite HeaRT using much fewer negative samples than the original setting. This suggests that the negative samples generated by HeaRT are substantially more challenging than those used in the existing setting. This underscores the need to develop more advanced methodologies that can tackle harder negatives samples like in HeaRT.

2.5 Conclusion

In this work we have revealed several pitfalls found in recent works on link prediction. To overcome these pitfalls, we first establish a benchmarking that facilitates a fair and consistent evaluation across a diverse set of models and datasets. By doing so, we are able to make several illuminating observations about the performance and characteristics of various models. Furthermore, based on several limitations we observed in the existing evaluation procedure, we introduce a more practical setting called HeaRT (Heuristic Related Sampling Technique). HeaRT incorporates a more real-world evaluation setting, resulting in a better comparison among methods. By introducing a more rigorous and realistic assessment, HeaRT could guide the field towards more effective models, thereby advancing the state of the art in link prediction.

CHAPTER 3

LPFORMER: AN ADAPTIVE GRAPH TRANSFORMER FOR LINK PREDICTION

3.1 Introduction

Link prediction (LP) attempts to predict unseen edges in a graph. It has been adopted in many applications including recommender systems [46], social networks [27], and drug discovery [1]. Traditionally, hand-crafted heuristics were used to identify new links in the graph [81, 151, 2]. Heuristics are often chosen based on factors that typically correlate well with the formation of new links. For example, a popular heuristic is common neighbors (CNs), which assume that the links are more likely to exist between node pairs with more shared neighbors. It has been found that these factors, which we refer to as "LP Factors", often stem from the local and global structural information and feature proximity [72]. We give an example in Figure 4.1 that demonstrates different heuristic scores for multiple candidate links. Each heuristic score corresponds to one of the LP factors: CNs for local information, Katz for global, and Feat-Sim for feature proximity. We can observe that the pair (source, 5) has the highest CN and Katz score of the candidate links, indicating an abundance of local and global structural information between the pair. On the other hand, the feature similarity for (source, 5) is the lowest among the candidate links. This indicates that *different LP factors and heuristics have distinct assumptions about why links are formed*.

More recently, message passing neural networks (MPNNs) [37], which are able to learn effective node representations via message passing, have been widely adopted for LP tasks. They predict the existence of a link by combining the node representations of both nodes in the link. However, such a node-centric view is unable to incorporate the pairwise information between the nodes in the link. Because of this, conventional MPNNs have been demonstrated to be poor link predictors due to their limited capability to learn effective and expressive link representations [142, 102]. To address this issue, recent efforts [140, 154] have attempted to move beyond the node-centric view of traditional MPNNs by equipping them with pairwise information specific to the link being predicted (i.e. the "target link") [140, 154]. This is done by customizing the message passing process to each

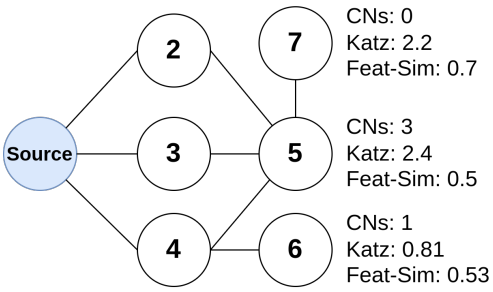


Figure 3.1 Example of multiple heuristic scores for the candidate links (source, 5), (source, 6), and (source, 7). Each heuristic corresponds to a different LP factor – local (CNs), global (Katz), and feature proximity (Feat-Sim).

target link. However, a concern with this approach is that it can be prohibitively expensive [20], as message passing needs to be run for each individual target link. This is as opposed to traditional MPNNs which only run message passing once for all target links.

To overcome these inefficiencies, recent methods [136, 20, 121] have instead explored ways to inject pairwise information into the model, without individualizing the message passing to each target link. This is done by decoupling the message passing and link-specific pairwise information. By doing so, the message passing only needs to be done once for all target links. To include the pairwise information, these methods, which we refer to as "Decoupled Pairwise MPNNs" (DP-MPNNs), instead learn a "pairwise encoding" to encode the pairwise relationship of the target link. The choice of pairwise encoding is often based on heuristics that correspond to common LP factors (e.g., common neighbors). DP-MPNNs have gained attention as they can achieve promising performance while being much more efficient than methods that customize the message passing mechanism.

However, DP-MPNNs are often limited in the choice of pairwise encoding, using a one-size-fits-all solution for all target links. This has two limitations. **(1)** The pairwise encoding may fail to consider some integral LP factors. For example, NCNC [121] only considers the 1-hop neighborhood when computing the pairwise encoding, thereby ignoring the global structural information. *This suggests the need for a pairwise encoding that considers multiple types of LP factors*. **(2)** The pairwise encoding uses the same LP factors for all target links. This assumes that all target links need the same factors. However, it may not necessarily be true. Recently, [72] have

shown that different LP factors are necessary to classify different target links. It is evident that even for the same dataset, multiple LP factors are needed to properly predict all target links. This further applies to different datasets, where certain factors are more prominent than others. As such, it faces tremendous challenges when considering multiple types of LP factors. While one factor may effectively model some target links, it will fail for other target links where those patterns aren't present. *It is therefore desired to consider different LP factors for different target links*.

These observations motivate us to ask – *can we design an efficient method that can adaptively determine which LP factors to incorporate for each individual target link?* Essentially, it requires a pairwise encoding that (a) models multiple LP factors, (b) can be tailored to fit each individual target link, and (c) is efficient to calculate. By doing so, we can flexibly adapt the pairwise information based on the existing needs of each target link. To achieve this, we propose **LPFormer** – **L**ink **P**rediction Trans**Former**. LPFormer is a type of graph Transformer [78] designed specifically for link prediction. Given a target link (a, b), LPFormer models the pairwise encoding via an attention module that learns how a and b relate in the context of various LP factors. This allows for a more customizable set of pairwise encodings that are specific to each target link. Extensive experiments validate that LPFormer can achieve SOTA on a variety of benchmark datasets. We further demonstrate that LPFormer is better at modeling several types of LP factors, highlighting its adaptability, while also maintaining efficiency on denser graphs.

3.2 Background

3.2.1 Related Work

Link prediction (LP) aims to model how links are formed in a graph. The process by which links are formed, i.e., link formation, is often governed by a set of underlying factors [8, 65]. We refer to these as "LP factors". Two categories of methods are used for modeling these factors – heuristics and MPNNs. We describe each class of methods. We further include a discussion on existing graph transformers.

Heuristics for Link Prediction. Heuristics methods [81, 151] attempt to explicitly model the LP factors via hand-crafted measures. Recently, [72] have shown that there are three main

factors that correlate with the existence of a link: (1) local structural information, (2) global structural information, and (3) feature proximity. **Local structural information** only considers the immediate neighborhood of the target link. Representative methods include Common Neighbors (CN) [81], Adamic Adar (AA) [2], and Resource Allocation (RA) [151]. They are predicated on the assumption that nodes that share a greater number neighbors exhibit a higher probability of forming connections. **Global structural information** further considers the global structure of the graph. Such methods include Katz [49] and Personalized PageRank (PPR) [15]. These methods posit that nodes interconnected by a higher number of paths are deemed to have larger similarity and, therefore, are more likely to form connections. Lastly, **feature proximity** assumes nodes with more similar features connect [79]. Previous work [83, 145] have shown that leveraging the node features are helpful in predicting links. Lastly, we note that [72] has recently shown that *to properly predict a wide variety of links, it's integral to incorporate all three of these factors*.

MPNNs for Link Prediction. Message Passing Neural Networks (MPNNs) [37] aim to learn node representations via the message passing mechanism. Traditional MPNNs have been used for LP including GCN [53], SAGE [42], and GAE [54]. However, they have been shown to be suboptimal for LP as they aren't expressive enough to capture important pairwise patterns [141, 102]. SEAL [140] and NBFNet [154] try to address this by customizing the message passing process to each target link. This allows for the message passing to learn pairwise information specific to the target link. However, these methods have been shown to be unduly expensive as they require a separate round of message passing for each target link. As such, recent methods have been proposed to instead decouple the message passing and pairwise information [136, 20, 121], reducing the time needed to do message passing. Such methods include NCN/NCNC [121] which exploit the common neighbor information and BUDDY [20] and Neo-GNN [136] which consider the global structural information.

Graph Transformers. Recent work has attempted to extend the original Transformer [114] architecture to graph-structured data. Graphormer [133] learns node representations by attending all nodes to each other. To properly model the structural information, they propose to use multiple

types of structural encodings (i.e., structural, centrality, and edge). SAN [58] further considers the use of the Laplacian positional encodings (LPEs) to enhance the learnt structural information. Alternatively, TokenGT [51] considers all nodes and edges as tokens in the sequence when performing attention. Due to the large complexity of these models, they are unable to scale to larger graphs. To address this, several graph transformers [22, 124] have been proposed for node classification that attempt to efficiently attend to the graph. However, while some work [23, 85] have formulated transformers for knowledge graph completion, to our knowledge, **there are no graph transformers designed specifically for LP on uni-relational graphs**.

3.2.2 Preliminaries

We denote a graph as $\mathcal{G} = \{\mathcal{V}, \mathcal{E}\}$, where $\mathcal{V}$ and $\mathcal{E}$ are the sets of nodes and edges in $\mathcal{G}$, respectively. The adjacency matrix is represented as $A \in \mathbb{R}^{|V|\times|V|}$. The d-dimensional node features are represented by the matrix $X \in \mathbb{R}^{|V|\times d}$. The set of neighbors for a node v is given by $\mathcal{N}(v)$. The set of overlapping neighbors between two nodes a and b, i.e., the common neighbors (CNs), is expressed by $\mathcal{N}^{\text{CN}}_{(a,b)}$. We further denote the set of nodes that are 1-hop neighbors of only one of a or b as $\mathcal{N}^{1}_{(a,b)}$ and the nodes that are >1-hop from both nodes as $\mathcal{N}^{>1}_{(a,b)}$. Lastly, the personalized pagerank (PPR) score for a root node v and an arbitrary node u is given by $\text{ppr}(v, u)$.

3.3 The Proposed Framework

In Section 5.1, we highlighted the importance of adaptively modeling multiple types of LP factors. However, current methods that use pairwise encodings, i.e., DP-MPNNs, struggle to appropriately achieve this goal. This is due to two issues: **(1)** They only attempt to model a subset of the potential LP factors (e.g., only local structural information), limiting their ability to model multiple factors. **(2)** They use a one-size-fits-all approach in regard to pairwise encoding, using the same combination of LP factors for each target link. These issues strongly limit the potential of such methods to properly model a variety of different target links. To overcome these problems, we propose **LPFormer**, a new transformer-based method that can adaptively customize the pairwise information for each target link by considering a variety of different LP factors in an efficient manner.

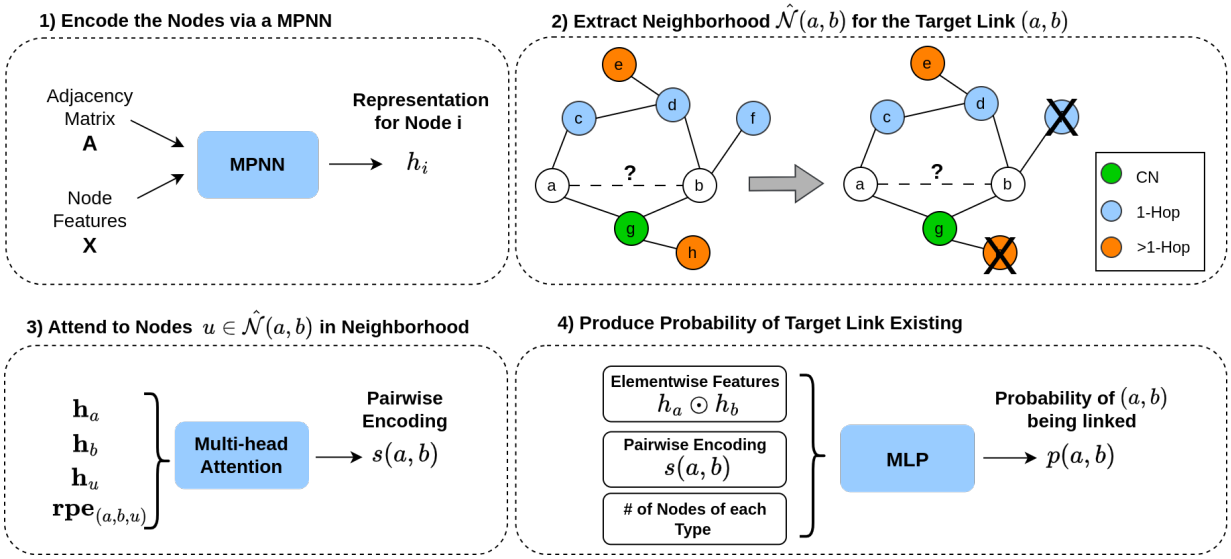


Figure 3.2 An overview of LPFormer. **(1)** Encode the nodes via a MPNN. **(2)** For a given target link, we determine which nodes to attend to ($\hat{\mathcal{N}}(a,b)$) via the PPR-based thresholding technique in Eq. (3.10). **(3)** The pairwise encoding is computed by attending to each node, $u \in \hat{\mathcal{N}}(a,b)$ using the feature and relative positional encoding $\text{rpe}_{(a,b,u)}$. **(4)** The pairwise encoding, node representations, and counts of different node types are concatenated and used to compute the final probability of the target link existing.

3.3.1 A General View of Pairwise Encodings

Recent MPNNs for LP use a decoupled strategy to include the pairwise information [20, 121, 136]. These methods, DP-MPNNs, predict the existence of a link (a,b) via both the node representations and a pairwise encoding $s(a,b)$. They follow the formulation below:

$$
\begin{aligned}
H &= \text{MPNN}(A, X), \\
p(a,b) &= \sigma\left(\text{MLP}\left(\mathbf{h}_a \odot \mathbf{h}_b \,\middle\|\, s(a,b)\right)\right),
\end{aligned}
\tag{3.1}
$$

where h_i is the representation of node i encoded by the MPNN. Various DP-MPNNs adopt different ways to model the pairwise encoding. For example, NCN [121] models the pairwise encoding $s(a,b)$ as the summation of the node representations of the CNs. The definitions of $s(a,b)$ for other prominent DP-MPNNs can be found in Appendix B.1. The pairwise encodings in these existing methods are typically manually selected or extracted from the graph, which limits the LP factors they can cover. For example, $s(a,b)$ in NCN and NCNC only capture the local structural information. BUDDY [20] ignores the node features when computing the pairwise encoding.

To flexibly model multiple types of LP factors, we propose a general formulation for pairwise encodings as follows,

$$s(a,b) = \sum_{u \in \mathcal{V}} w(a,b,u) \odot h(a,b,u), \tag{3.2}$$

where $w(a,b,u)$ measures the importance of node u to (a,b), and $h(a,b,u)$ is the encoding of node u relative to (a,b). By considering *which* nodes should be considered for (a,b) and *how* they are related to the node pair, Eq. (3.2) can model different LP factors by manually defining $w(a,b,u)$ and $h(a,b,u)$. In particular, we demonstrate how the heuristic methods corresponding to different LP factors can fit into this framework.

Common Neighbors (CNs) [81]: CNs considers the local structural information and is defined for a pair of nodes (a,b) as $\mathcal{N}^{\text{CN}}_{(a,b)} = \mathcal{N}(a) \cap \mathcal{N}(b)$. Eq. (3.2) is equal to the CNs when $h(a,b,u) = 1$ and:

$$w(a,b,u) = \left\{ \begin{array}{ll} 1, & \text{when } u \in \mathcal{N}(a) \cap \mathcal{N}(b) \\ 0, & \text{else} \end{array} \right\}. \tag{3.3}$$

Katz Index [49]: The Katz index models the global structural information. It is defined as weighted summation of the number of paths of different lengths connecting a and b and a decay weight $\beta \in [0,1]$,

$$\text{Katz}(a,b) = \sum_{l=1}^{\infty} \beta^l A^l_{a,b}.$$

This is equivalent to Eq. (3.2) where $w(a,b,u) = \sum_{l=1}^{\infty} \beta^l e_a^T A^l$ and

$$h(a,b,u) = \left\{ \begin{array}{ll} e_b^T, & \text{when } u = b \\ \mathbf{0}, & \text{else} \end{array} \right\},$$

where $e_i \in \mathbb{B}^{|\mathcal{V}|}$ is a one-hot vector for a node i.

Feature Similarity: The feature similarity of the pair of nodes (a,b) is expressed by $\text{dis}(\mathbf{x}_a, \mathbf{x}_b)$ where $\mathbf{x}_a$ are the node features of node a and $\text{dis}(\cdot)$ is a distance function (e.g., euclidean distance). This can be rewritten as Eq. (3.2) by substituting $w(a,b,u) = \text{dis}(\mathbf{x}_a, \mathbf{x}_u)$ and $h(a,b,u) = e_b^T$.

These examples demonstrate that the general formulation can indeed model many different LP factors including local and global structural information and feature proximity. We further show

in Appendix B.2 that Eq. (3.2) can model a variety of additional LP factors including RA [151], the pairwise encodings used in NCN/NCNC [121] and Neo-GNN [136]. However, fitting these methods into the formulation in Eq. (3.2) requires manually defining both $w(a, b, u)$ and $h(a, b, u)$. This constrains the information represented by $s(a, b)$ based on the choice of design. Motivated by this, in the next section we introduce our method that does not rely on a handcrafting both $w(a, b, u)$ and $h(a, b, u)$.

3.3.2 Modeling Pairwise Encodings via Attention

In Section 3.3.1, we introduced a general formulation for pairwise encodings in Eq. (3.2), which is able to capture a variety of different LP factors. However, it requires manually defining both terms in the equation. This limits our ability to customize the pairwise information to each target link. As such, we further aim to move beyond a one-size-fits-all pairwise encoding, and enable the model to produce customized pairwise encoding for each target link. This allows the model to handle more realistic graphs that often contain multiple prominent LP factors for different target links as shown in [72].

In particular, we consider the following question: *How can we model Eq* (3.2) *such that it can customize the used LP factors to each target link?* We consider parameterizing both $w(a, b, u)$ and $h(a, b, u)$. This allows us to learn how to personalize them to each target link. To achieve this, we leverage softmax attention [6]. This is due to its ability to dynamically learn the relevance of different nodes to the target link. As such, for multiple target links, it can emphasize the contributions of different nodes, thereby flexibly modeling different LP factors. We note that since the attention is between different sequences (i.e., a target link and nodes), it can be considered a form of cross attention [114].

To enhance the adaptability of the pairwise encoding for various links, it is essential to incorporate various types of information. This allows the attention mechanism to discern and prioritize relevant information for each target link, facilitating the effective modeling of diverse LP factors. In particular, we consider two types of information. The first is the **feature information**. This includes the feature representation of both nodes in the target link and the node being attended

to. The node features are included due to their role in link formation and relationship to structural information [79]. Second, we consider the **relative positional information**. The relative positional information reflects the relative position in the graph of a node u to the target link (a, b) in the local and global structural context. Due to the importance of local and global structural information [31, 45], it is vital to properly encode both. By including both the structural and feature information, we are able to cover the space of potential LP factors (see Section 3.2.1).

We denote the feature representation of a node u as $\mathbf{h}_u$ and the relative positional encoding (RPE) as $\mathbf{rpe}_{(a,b,u)}$. The node importance $w(a, b, u)$ is modeled via attention as follows:

$$\begin{aligned} \tilde{w}(a, b, u) &= \phi\left(\mathbf{h}_a, \mathbf{h}_b, \mathbf{h}_u,\ \mathbf{rpe}_{(a,b,u)}\right), \\ w(a, b, u) &= \frac{\exp(\tilde{w}(a, b, u))}{\sum_{v \in \bar{\mathcal{V}}(a,b)} \exp(\tilde{w}(a, b, u))}, \end{aligned} \tag{3.4}$$

where $\bar{\mathcal{V}}(a, b) = \mathcal{V} \setminus \{a, b\}$. The attention weight $w(a, b, u)$ can be considered as the impact of a node u on (a, b) relative to all nodes in $\mathcal{G}$. This allows the model to emphasize different LP factors for each target link. The node encoding $h(a, b, u)$ includes the features of node u in conjunction with the RPE and is defined as:

$$h(a, b, u) = \mathbf{W}\left[\mathbf{h}_u \ \Big\| \ \mathbf{rpe}_{(a,b,v)}\right]. \tag{3.5}$$

By substituting Eq. (3.4) and Eq. (3.5) into Eq. (3.2) we can compute the pairwise information $s(a, b)$. We further define $\phi(\cdot)$ in Eq. (3.4) as the GATv2 [17] attention mechanism. The detailed formulation is given in Appendix B.4. The feature representations $\mathbf{h}_i$ are computed via a MPNN. We use GCN [55] in this work. However, it is unclear how to properly encode the RPE of a node u relative to (a, b), $\mathbf{rpe}_{(a,b,u)}$. We aim to design the RPE to capture both the local and global structural relationship between the node and target link while also being efficient to calculate. In the next section, we discuss our solution for modeling $\mathbf{rpe}_{(a,b,u)}$.

3.3.3 PPR-Based Relative Positional Encodings

In this section, we introduce our strategy for computing the RPE of a node u relative to a target link (a, b). Intuitively, we want the RPE to reflect the positional relationship between u and

(a, b) such that different types of information (i.e., local vs. global) are encoded differently. Using Figure 4.1 as an example, since node 3 is a CN of (source, 5) we expect it to have a much different relationship to the target link than node 6, which is a 2-hop neighbor of both nodes. An enticing option is to use the double radius node labeling (DRNL) trick introduced by [140]. However, [20] have shown it to be prohibitively expensive to calculate for larger graphs. Furthermore, existing RPEs are typically infeasible to calculate on larger graphs as they often rely on pairwise distances or the eigenvectors of the Laplacian [88].

As such, we seek an RPE that can both distinguish the relationship of different nodes to the target link while also being efficient to calculate. To motivate our RPE design, we draw inspiration from the following Proposition.

Proposition 1. *Consider a target link (a, b) and a node $u \in \mathcal{V} \setminus \{a, b\}$. The PPR [15] score of a root node i and target node j with teleportation probability α is denoted by $ppr(i, j)$. Let $r_a^k(u)$ be the probability of a walk of length k beginning at node a and terminating at u. We define $r_{a,b}^k(u) := r_a^k(u) + r_b^k(u)$. We also define a weight $\gamma^k := \alpha(1-\alpha)^k$ for all walks of length k. The PPR scores, $ppr(a, u)$ and $ppr(b, u)$, along with the random walk probabilities of disparate lengths, are interconnected through the following relationship.*

$$\Gamma(a, b, u) = ppr(a, u) + ppr(b, u) = \sum_{k=0}^{\infty} \gamma^k r_{a,b}^k(u). \tag{3.6}$$

The detailed proof is given in Appendix B.3. From Proposition 1, we can make the following observations: (1) The PPR scores encode the weighted sum of the probabilities of different length random walks connecting two nodes. (2) Walks of shorter length are given higher importance, as evidenced by the dampening factor $\gamma^k = \alpha(1-\alpha)^k$ which decays with the increase in k. These observations imply that – **a larger value of $\Gamma(a, b, u)$ correlates with the existence of many shorter walks connecting node u to the both nodes in the target link (a, b).**

Therefore, the PPR scores can be used as an intuitive and useful method to understand the structural relationship between node u and both nodes in the target link (a, b). If both scores, $\text{ppr}(a, u)$ and $\text{ppr}(b, u)$, are high, there exists a high probability that many shorter walks connect

u to both nodes in the target link. This implies that node u has a stronger impact on the nodes in the target link. On the other hand, if both PPR scores are low, there is likely very little relationship between u and the target link. This allows for a convenient way of differentiating how a node structurally relates to the target link. Furthermore, we note that the PPR matrix can be efficiently pre-computed using the algorithm introduced by [5], allowing for easy computation and use.

Following this idea, to calculate the RPE of a node u, we use the PPR scores of a node u relative to both nodes in the target link (a, b). Instead of considering the sum of PPR scores as in Proposition 1, we further parameterize $\Gamma(\cdot)$ via an MLP,

$$\mathbf{rpe}_{(a,b,u)} = \text{MLP}\left(\text{ppr}(a, u), \text{ppr}(b, u)\right). \tag{3.7}$$

By introducing learnable parameters to $\Gamma(\cdot)$, it allows for the model learn the importance of individual PPR scores and how they interact with each other. To ensure that Eq. (3.7) is invariant to the order of the nodes in the target link, i.e., (a, b) and (b, u), we further set the RPE to be equal to the summation of the representations given by both (a, b) and (b, a):

$$\overline{\mathbf{rpe}}_{(a,b,u)} = \mathbf{rpe}_{(a,b,u)} + \mathbf{rpe}_{(b,a,u)}. \tag{3.8}$$

However, a concern with Eq. (3.8) is that it is not guaranteed to be able to distinguish certain types of nodes from each other. For example, it is necessary to clearly distinguish CNs from other nodes due to their important role in link formation [81]. To overcome this issue, we fit three separate MLPs for when u is a: CN of (a, b), a 1-hop neighbor of either a and b, and a >1-hop neighbor of both a and b. This ensures that we can properly distinguish between these three types of nodes. We verify the effectiveness of this design in Section 3.4.4. Lastly, we note that while other work [75, 63] has considered the use of random-walk based positional encodings, they are only designed for use on the node-level and are unable to be used for link-level tasks like LP.

3.3.4 Efficiently Attending to the Graph Context

The proposed attention mechanism in Section 3.3.2 attends to all nodes in the graph, sans those in the link itself. This makes it difficult to scale to large graphs. Motivated by selective [73] and sparse [25] attention, we opt to attend to only a small portion of the nodes.

At a high level, we are interested in determining a subset of nodes $\hat{\mathcal{N}}(a,b) \in \mathcal{V}$ to attend to for the target link (a,b). Our goal is to choose the set of nodes $\hat{\mathcal{N}}(a,b)$ such that they are **(a)** few in number to improve scalability and **(b)** provide important contextual information to the pair (a,b) to best learn the pairwise information. This can be achieved by only considering all nodes where the importance of the node u to the target link (a,b) is considered high. Formally, we can write this as the following where $\mathcal{I}(a,b,u)$ is a function that denotes the importance of a node u to the target link (a,b):

$$\hat{\mathcal{N}}(a,b) = \{u \in \mathcal{V} \setminus \{a,b\} \mid \mathcal{I}(a,b,u) > \eta\}. \tag{3.9}$$

The threshold η allows us to distinguish those nodes that are sufficiently important to the target link. This allows for a simple and efficient way of determining the set $\hat{\mathcal{N}}(a,b)$. However, *what do we use to model the importance $\mathcal{I}(a,b,u)$?* For ease of optimization and better efficiency, we avoid parameterizing the function $\mathcal{I}(a,b,u)$. Instead, we want to choose a metric such that can properly serve as a proxy for the importance of a node u to (a,b) while also being concentrated in a small subset of nodes. Such a metric will allow Eq. (3.9) to choose a small but influential set of nodes to attend to.

A measure that satisfies both criteria is Personalized Pagerank (PPR) [15]. In Section 3.3.3 we discussed that the PPR score can serve as a good tool to model the influence of a one node on another. Furthermore, existing work [38, 80, 5] shows that the PPR scores tend to be highly localized in a small subset of nodes. Therefore by making $\mathcal{I}(a,b,u)$ contingent on the PPR scores of (a,u) and (b,u) we can extract a small but important set of nodes to attend to for the target link.

Following this idea, for a target link (a,b), we keep all nodes whose PPR score is above some threshold η relative to both nodes in the target link. As such, we only keep a node u if it is related in some capacity to at least one of the nodes in the target link. Similarly to Section 3.3.3, we treat CN, 1-Hop, and >1-Hop nodes differently by applying a different threshold for them. The filtered node set for each category of nodes is given by:

$$\hat{\mathcal{N}}^{\pi}_{(a,b)} = \{u \in \mathcal{N}^{\pi}_{(a,b)} \mid \text{ppr}(a,u) > \eta^{\pi},\ \text{ppr}(b,u) > \eta^{\pi}\}, \tag{3.10}$$

Table 3.1 Dataset statistics. The split ratio is the % of samples for train/validation/test.

	Cora	Citeseer	Pubmed	ogbl-collab	ogbl-ddi	ogbl-ppa	ogbl-citation2
#Nodes	2,708	3,327	18,717	235,868	4,267	576,289	2,927,963
#Edges	5,278	4,676	44,327	1,285,465	1,334,889	30,326,273	30,561,187
Split Ratio	85/5/10	85/5/10	85/5/10	92/4/4	80/10/10	70/20/10	98/1/1

where $\hat{\mathcal{N}}^{\pi}_{(a,b)}$ is the filtered node set for all nodes of the type $\pi \in \{\text{CN}, 1-\text{Hop}, >1-\text{Hop}\}$ and η^{π} is the corresponding PPR threshold. We note that while other work [10, 134] has used PPR to filter the nodes on the *node-level*, no existing work has done so on the *link-level*.

We corroborate this design by demonstrating that LPFormer can achieve SOTA performance in LP (Section 3.4.2) while achieving a faster runtime than the second-best method, NCNC [121], on denser graphs (Section 3.4.7). This is despite the fact that LPFormer can attend to a wider variety of nodes. We further show in Section 3.4.5 that the performance is stable with regards to the values of η chosen, allowing us to easily choose a proper threshold on any dataset.

3.3.5 LPFormer

We now define the overall framework – LPFormer. The overall procedure is given in Figure 3.2: **(1)** We first learn node representations from the input adjacency and node features via an MPNN. We note that this step is agnostic to the target link. **(2)** For a target link (a, b) we extract the nodes to attend to, i.e. $\hat{\mathcal{N}}(a, b)$. This is done via the PPR thresholding technique defined in Section 3.3.4. **(3)** We apply L layers of attention, using the mechanism defined in Section 3.3.2. The output is the pairwise encoding $s(a, b)$. **(4)** We generate the prediction of the target link using three types of information: the element-wise product of the node representation, the pairwise encoding, and the number of CN, 1-Hop, and >1-Hop nodes identified by Eq. (3.10). The score function is given by:

$$p(a,b) = \sigma\left(\text{MLP}\left(\mathbf{h}_a \odot \mathbf{h}_b \,\middle\|\, s(a,b) \,\middle\|\, |\hat{\mathcal{N}}^{\text{CN}}_{(a,b)}| \,\middle\|\, |\hat{\mathcal{N}}^{1}_{(a,b)}| \,\middle\|\, |\hat{\mathcal{N}}^{>1}_{(a,b)}|\right)\right) \tag{3.11}$$

We demonstrate in Section 3.4.4 that the inclusion of the node counts is helpful, as it provides complementary information to the pairwise encoding.

3.4 Experiments

In this section, we conduct extensive experiments to validate the effectiveness of LPFormer. Specifically, we attempt to answer the following questions: **(RQ1)** Can LPFormer consistently

outperform baseline methods on a variety of different benchmark datasets? **(RQ2)** Is LPFormer able to model a variety of different LP factors? **(RQ3)** Can LPFormer be run efficiently on large dense graphs? We further conduct studies ablating each component of our model and analyzing the effect of the PPR-based threshold on performance.

3.4.1 Experimental Settings

Datasets. We include Cora, Citeseer, and Pubmed [131] and ogbl-collab, ogbl-ppa, ogbl-ddi, and ogbl-citation2 [44]. Furthermore, for Cora, Citeseer, and Pubmed we experiment under a single fixed split (see Appendix B.5.1 for further discussion). The detailed statistics for each dataset are shown in Table A.1.

Baseline Models. We compare LPFormer against a wide variety of baselines including: CN [81], AA [2], RA [151], GCN [55], SAGE [42], GAE [54], SEAL [140], NBFNet [154], Neo-GNN [136], BUDDY [20], and NCNC [121]. Results on Cora, Citeseer, and Pubmed are taken from [61]. Results for the heuristic methods are from [44]. All other results are either from their respective study or [20].

Hyperparameters: The learning rate is tuned from $\{1e^{-3}, 5e^{-3}\}$, the decay from $\{0.95, 0.975, 1\}$, and the dropout from $[0, 0.7]$, and the weight decay from $\{0, 1e^{-4}, 1e^{-7}\}$. The size of the hidden dimension is set to 64 for ogbl-ppa and ogbl-citation2, 128 for Cora, Pubmed, and ogbl-collab, and 256 for Citeseer. Lastly, the PPR threshold is tuned from $\{1e^{-2}, 1e^{-3}, 1e^{-4}\}$.

Evaluation Metrics. Each positive target link is evaluated against a set of given negative links. The rank of the positive link among the negatives is used to evaluate performance. The two types of metrics that are used to evaluate this ranking are Hits@K and MRR. For the OGB datasets we use the metric used in the original study. This includes Hits@50 for ogbl-collab, Hits@100 for ogbl-ppa and MRR for ogbl-citation2. For Cora, Citeseer, Pubmed we follow [61] and use MRR. Lastly, the same set of negative links is used for all positive links except on ogbl-citation2, where [44] provides a customized set of 1000 negatives for each individual positive link.

Table 3.2 Results on benchmark datasets. OOM is an out of memory error. We highlight the results ranked first, second, and third as green, blue, and orange, respectively.

	Cora	Citeseer	Pubmed	ogbl-collab	ogbl-ppa	ogbl-citation2	Mean Rank
Metric	MRR	MRR	MRR	H@50	H@100	MRR	
CN	20.99±0.00	28.34±0.00	14.02±0.00	56.44±0.00	27.65±0.00	51.47±0.00	11.0
AA	31.87±0.00	29.37±0.00	16.66±0.00	64.35±0.00	32.45±0.00	51.89±0.00	8.5
RA	30.79±0.00	27.61±0.00	15.63±0.00	64.00±0.00	49.33±0.00	51.98±0.00	8.7
GCN	32.50±6.87	50.01±6.04	19.94±4.24	44.75±1.07	18.67±1.32	84.74±0.21	8.0
SAGE	**37.83**±7.75	47.84±6.39	22.74±5.47	48.10±0.81	16.55±2.40	82.60±0.36	7.7
GAE	29.98±3.21	**63.33**±3.14	16.67±0.19	OOM	OOM	OOM	NA
SEAL	26.69±5.89	39.36±4.99	**38.06**±5.18	64.74±0.43	48.80±3.16	87.67±0.32	6.2
NBFNet	**37.69**±3.97	38.17±3.06	**44.73**±2.12	OOM	OOM	OOM	NA
Neo-GNN	22.65±2.60	53.97±5.88	31.45±3.17	57.52±0.37	49.13±0.60	87.26±0.84	7.0
BUDDY	26.40±4.40	59.48±8.96	23.98±5.11	**65.94**±0.58	49.85±0.20	87.56±0.11	**5.7**
NCN	32.93±3.80	54.97±6.03	35.65±4.60	64.76±0.87	**61.19**±0.85	**88.09**±0.06	**3.8**
NCNC	29.01±3.83	**64.03**±3.67	25.70±4.48	**66.61**±0.71	**61.42**±0.73	**89.12**±0.40	**3.8**
LPFormer	**39.42**±5.78	**65.42**±4.65	**40.17**±1.92	**68.14**±0.51	**63.32**±0.63	**89.81**±0.13	**1.2**

3.4.2 Main Results

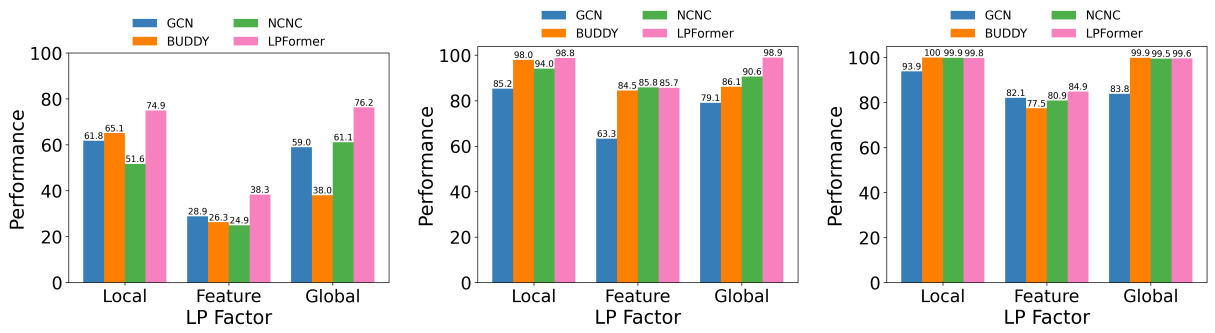
We present the results of LPFormer compared with baselines on multiple benchmark datasets. Note that we omit ogbl-ddi from the main results due to recent issues discovered by [61] (see Appendix B.5.2 for more details). The results are shown in Table 4.1. We observe that LPFormer can achieve SOTA performance on 5/6 datasets, significantly outperforming other baselines. Moreover, LPFormer is also the most consistent of all the methods, achieving strong performance on all datasets. This is as opposed to previous SOTA methods, NCNC and BUDDY, which tend to struggle on Cora and Pubmed. We attribute the consistency of LPFormer to the flexibility of our model, allowing it to customize the LP factors needed to each link and dataset.

3.4.3 Performance by LP Factor

In this section, we measure the ability of LPFormer to capture a variety of different LP factors. To measure this, we identify all positive target links **when there is only one dominant LP factor**. For example, one group would contain all target links where the only dominant factor is the local structural information. We focus on links that correspond to one of the three groups identified in [72]: local structural information, global structural information, and feature proximity.

We identify these groups by using popular heuristics as proxies for each factor. For local structural information, we use CNs [81], for global structural information we use PPR [15] as it's the most computationally efficient of all global methods, and for feature proximity, we use the cosine similarity of the features. Using these heuristics, we determine if only one factor is dominant by comparing the relative score of each heuristic. This is done by first computing the score for each factor i for the target link (a, b) – $s^i(a, b)$. For each factor, we then compute the score corresponding to the p-th percentile among all links, $\hat{s}^i$. We choose a larger value of p (i.e. 90%) such that a score $\geq \hat{s}^i$ indicates that a significant amount of pairwise information exists for that factor. For a single target link, we then compare the score of each factor $s^i(a, b)$ to $\hat{s}^i$. If $s^i(a, b) \geq \hat{s}^i$ is true **for only one factor**, this implies that the score for only one factor is "high". Therefore there is a notable amount of pairwise information existing for only one factor for the link (a, b). This ensures that only one factor is strongly expressed. If this is true, we then assign the target link (a, b) to factor i. Please see Appendix B.5.4 for a more detailed explanation.

We demonstrate the results on Cora, Citeseer, and ogbl-collab in Figure 3.3. We observe that LPFormer typically performs best for each individual LP factor on all datasets. Furthermore, it is also the most consistently well-performing on each factor as compared to other methods. For example, on Cora the other methods struggle for links that correspond to the feature proximity factor. LPFormer, on the other hand, is able to significantly outperform them on those target links, performing around 33% better than the second best method. Lastly, we note that most methods tend to perform well on the links corresponding to the global factor, even if they don't explicitly model such information. This is caused by a strong correlation that tends to exist between local and global structural information, often resulting in considerable overlap between both factors [72]. These results show that LPFormer can indeed adapt to multiple types of LP factors, as it can consistently perform well on samples belonging to a variety of different LP factors. Additional results are given in Appendix B.5.5.



(a) Cora (b) Citeseer (c) ogbl-collab

Figure 3.3 Performance on links that contain one dominant LP factor. Results are on (a) Cora, (b) Citeseer, and (c) ogbl-collab.

3.4.4 Ablation Study

We further include an ablation study to verify the effectiveness of the proposed components in LPFormer. In particular, we introduce 6 variants of LPFormer. (a) **w/o Learnable Att**: No attention is learned. As such, we set all attention weights to 1 and remove the RPE. (b) **w/o Features in Att**: We remove the node feature information from the attention mechanism. (c) **w/o RPE in Att**: We remove the RPE from the attention mechanism. (d) **w/o PPR RPE**: We replace the PPR-based RPE with a learnable embedding for each of CN, 1-Hop, and >1-Hop nodes. (e) **w/o PPR RPE by Node Type**: We don't fit a separate function for each node type when determining the PPR RPE (see Section 3.3.3). Instead we use one for all nodes. (f) **w/o Counts**: We remove the counts of different nodes from the scoring function.

The results are shown in Table 4.4. We include ogbl-collab, ogbl-ppa, and Citeseer. We observe that ablating a component always decreases the performance. However, the magnitude of the decrease is dataset-dependent. For example, on ogbl-collab, ablating the feature information in the attention marginally affects the performance. However, on ogbl-ppa and Citeseer, removing the feature information results in a large decrease in performance. On the other hand, while removing learnable attention results in a modest decrease on ogbl-ppa, for the other two datasets we see a large drop. This highlights the importance of each component of our framework, as they are each necessary for consistently strong performance across multiple datasets.

Table 3.3 Ablation Study on LPFormer.

Method	ogbl-collab	ogbl-ppa	Citeseer
w/o Learnable Att	65.05±0.50	62.77±1.03	56.23±1.75
w/o Features in Att	68.04±0.79	56.98±1.55	53.40±9.30
w/o RPE in Att	65.26±0.56	61.20±0.69	56.70±3.79
w/o PPR RPE	67.09±0.51	61.91±1.22	51.96±15.2
w/o PPR RPE by Node Type	67.95±0.54	62.92±1.06	57.40±5.71
w/o Counts	67.75±0.41	44.37±1.89	54.39±5.30
LPFormer	**68.14±0.51**	**63.32±0.63**	**65.42±4.65**

Table 3.4 Effect of Varying the PPR Thresholds.

Threshold	ogbl-collab		ogbl-citation2	
	1-Hop	>1−Hop	1-Hop	>1−Hop
1e-4	68.24±0.25	67.73±0.65	89.81±0.13	89.14±0.22
1e-2	67.60±0.31	68.24±0.25	89.49±0.18	89.81±0.13
1	67.08±0.65	68.14±0.51	89.49±0.16	89.26±0.39

3.4.5 Effect of the PPR Thresholds

We examine the effect of varying the PPR threshold for both 1-Hop and >1−Hop nodes as described in Eq. (3.10). The results for ogbl-collab and ogbl-citation2 are shown in Table 3.4. When varying the 1-Hop threshold, we fix the value of the >1−Hop threshold to 1e-2 for both datasets. When varying the >1−Hop threshold, we fix the value of the 1-Hop threshold to 1e-4 for both datasets.

We can observe that modifying the threshold has little effect on the underlying performance of the model. For both datasets, a value of 1e-2 works well for the >1−Hop threshold and 1e-4 works well for the 1-Hop threshold. We typically find that setting both values to 1e-2 provides a good trade-off between performance and efficiency.

3.4.6 Performance on HeaRT Setting

We further test the performance of our method on the HeaRT [61] evaluation setting, which considers a more realistic and difficult evaluation setting for link prediction. This is done by introducing a much harder and more realistic set of negative samples during evaluation. [61] observe that this results in a large decrease in performance on all datasets. Furthermore, compared to the

Table 3.5 Results (MRR) under HeaRT. We highlight the results ranked first, second, and third as green, blue, and orange, respectively.

Models	Cora	Citeseer	Pubmed	ogbl-collab	ogbl-ddi	ogbl-ppa	ogbl-citation2	Mean Rank
CN	9.78	8.42	2.28	4.20	6.71	25.70	17.11	11.1
AA	11.91	10.82	2.63	5.07	6.97	26.85	17.83	9.6
RA	11.81	10.84	2.47	**6.29**	8.70	28.34	17.79	8.1
GCN	**16.61 ± 0.30**	21.09 ± 0.88	7.13 ± 0.27	6.09 ± 0.38	**13.46 ± 0.34**	26.94 ± 0.48	19.98 ± 0.35	**4.7**
SAGE	14.74 ± 0.69	21.09 ± 1.15	**9.40 ± 0.70**	5.53 ± 0.5	12.60 ± 0.72	27.27 ± 0.30	**22.05 ± 0.12**	**4.7**
GAE	**18.32 ± 0.41**	**25.25 ± 0.82**	5.27 ± 0.25	OOM	3.49 ± 1.73	OOM	OOM	NA
SEAL	10.67 ± 3.46	13.16 ± 1.66	5.88 ± 0.53	**6.43 ± 0.32**	9.99 ± 0.90	29.71 ± 0.71	20.60 ± 1.28	6.4
NBFNet	13.56 ± 0.58	14.29 ± 0.80	>24h	OOM	>24h	OOM	OOM	NA
BUDDY	13.71 ± 0.59	22.84 ± 0.36	7.56 ± 0.18	5.67 ± 0.36	12.43 ± 0.50	27.70 ± 0.33	19.17 ± 0.20	5.9
Neo-GNN	13.95 ± 0.39	17.34 ± 0.84	7.74 ± 0.30	5.23 ± 0.9	10.86 ± 2.16	21.68 ± 1.14	16.12 ± 0.25	7.4
NCN	14.66 ± 0.95	**28.65 ± 1.21**	5.84 ± 0.22	5.09 ± 0.38	**12.86 ± 0.78**	**35.06 ± 0.26**	**23.35 ± 0.28**	**4.4**
NCNC	14.98 ± 1.00	24.10 ± 0.65	**8.58 ± 0.59**	4.73 ± 0.86	>24h	**33.52 ± 0.26**	19.61 ± 0.54	4.8
LPFormer	**16.80 ± 0.52**	**26.34 ± 0.67**	**9.99 ± 0.52**	**7.62 ± 0.26**	**13.20 ± 0.54**	**40.25 ± 0.24**	**24.70 ± 0.55**	**1.4**

original evaluation setting, MPNNs designed specifically for link prediction are often outperformed by heuristics or other MPNNs.

The full results can be found in Table 3.5. We observe that LPFormer performs considerably better than all other models. For instance, the mean rank of LPFormer is 3.1x better than the 2nd best-performing model, NCN. This indeed shows the advantage of LPFormer, as it can consistently achieve extraordinary performance across all datasets under the much more challenging HeaRT evaluation setting. This is as opposed to other LP-specific methods that often perform similarly to standard MPNN methods.

3.4.7 Runtime Analysis

In this section, we compare the runtime of LPFormer against NCNC, which is the strongest performing baseline. The results are shown in Figure 3.4 on all four OGB datasets We further include the mean degree of each dataset in parentheses. We observe that LPFormer shines on denser datasets, taking significantly less time to train one epoch. This is despite that LPFormer can attend to nodes beyond the 1-hop radius of the target link. This underscores the importance of the PPR thresholding technique introduced in Section 3.3.4, as it allows for efficient attention to a wider variety of nodes. Lastly, we note that LPFormer struggles on the ogbl-citation2 dataset due to the large number of nodes in the dataset (i.e., 2,927,963), which requires the sparse PPR matrix to be quite large. For future work we plan on exploring pre-computing the necessary PPR scores as

an efficient pre-processing step, thereby removing the need to store the costly PPR matrix. Please see Appendix B.5.7 for more details.

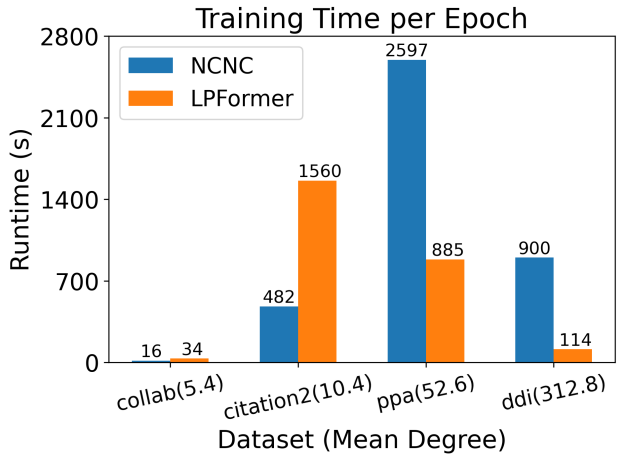


Figure 3.4 Comparison of training time of 1 epoch between LPFormer and NCNC. The mean degree is in parentheses.

3.5 Conclusion

In this paper we introduce a new framework, LPFormer, that aims to integrate a wider variety of pairwise information for link prediction. LPFormer does this via a specially designed graph transformer, which adaptively considers how a node pair relate to each other in the context of the graph. Extensive experiments demonstrate that LPFormer can achieve SOTA performance on a wide variety of benchmark datasets while retaining efficiency. We further demonstrate LPFormer's supremacy at modeling multiple types of LP factors. For future work, we plan on exploring other methods of incorporating multiple LP factors with an emphasis on global structural information. We also plan to investigate the potential of alternative relative positional encodings.

CHAPTER 4

TOWARD DEGREE BIAS IN EMBEDDING-BASED KNOWLEDGE GRAPH COMPLETION

4.1 Introduction

Knowledge graphs (KGs) are a specific type of graph where each edge represents a single fact. Each fact is represented as a triple (h, r, t) that connects two entities h and t with a relation r. KGs have been widely used in many real-world applications such as recommendation [18], drug discovery [77], and natural language understanding [67]. However, the incomplete nature of KGs limits their applicability in those applications. To address this limitation, it is desired to perform a KG completion (KGC) task, i.e., predicting unseen edges in the graph thereby deducing new facts [93]. In recent years, the embedding-based methods [13, 29, 7, 100] that embed a KG into a low-dimensional space have achieved remarkable success on KGC tasks and enable downstream applications.

However, a common issue in graph-related tasks is degree bias [107, 56], where nodes of lower degree tend to learn poorer representations and have less satisfactory downstream performance. Recent studies have validated this issue for various tasks on homogeneous graphs such as classification [107, 146, 68] and link prediction [56]. However, KGs are naturally heterogeneous with multiple types of nodes and relations. Furthermore, the study of degree bias on KGs is rather limited. Therefore, in this work, we ask *whether degree bias exists in KGs and how it affects the model performance in the context of KGC*.

To answer the aforementioned question, we perform preliminary studies to investigate how the degree affects the KGC performance. Take a triple (h, r, t) as one example. The in-degree of entity t is the number of triples where t is the tail entity. Furthermore, we define the *tail-relation degree* as the number of triples where t and the relation r co-occur as the tail and relation (Eq. (4.2)). An example in Figure 4.1 is the tail-relation pair (*Germany*, *Has Country*). Since the pair only co-occurs as a relation and tail in one triple, their tail-relation degree is 1. Our preliminary studies (Section 4.3) suggest that when predicting the tail entity t, the in-degree of t and especially the

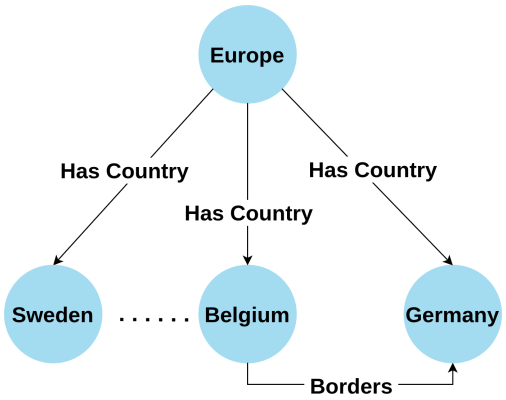


Figure 4.1 Example of multiple facts in a KG. Since each country only co-occurs with the relation *Has Country* as the tail on one edge, they each only have a tail-relation degree of one with the relation *Has Country*.

tail-relation degree of (t, r) plays a vital role. That is, when predicting the tail for a triple (h, r, t), the number of triples where the entity t and relation r co-occur as an entity-relation pair correlates significantly with performance during KGC. Going back to our example, since *Germany* and *Has Country* only co-occur as a relation and tail in one triple their tail-relation degree is low, thus making it difficult to predict *Germany* for the query (*Europe*, *Has Country*, ?).

Given the existence of degree bias in KGC, we aim to alleviate the negative effect brought by degree bias. Specifically, we are tasked with improving the performance of triples with low tail-relation degrees while maintaining the performance of other triples with a higher tail-relation degree. Essentially, it is desired to promote the engagement of triples with low tail-relation degrees during training so as to learn better embeddings. To address this challenge, we propose a novel data augmentation framework. Our method works by augmenting entity-relation pairs that have low tail-relation degrees with synthetic triples. We generate the synthetic triples by extending the popular Mixup [137] strategy to KGs. Our contributions can be summarized as follows:

- Through empirical study, we identify the degree bias problem in the context of KGC. To the best of our knowledge, *no previous work has studied the problem of degree bias from the perspective of entity-relation pairs*.
- We propose a simple yet effective data augmentation method, KG-Mixup, to alleviate the degree bias problem in KG embeddings.

- Through empirical analysis, we show that our proposed method can be formulated as a form of regularization on the low tail-relation degree samples.
- Extensive experiments have demonstrated that our proposed method can improve the performance of lower tail-relation degree triples on multiple benchmark datasets without compromising the performance on triples of higher degree.

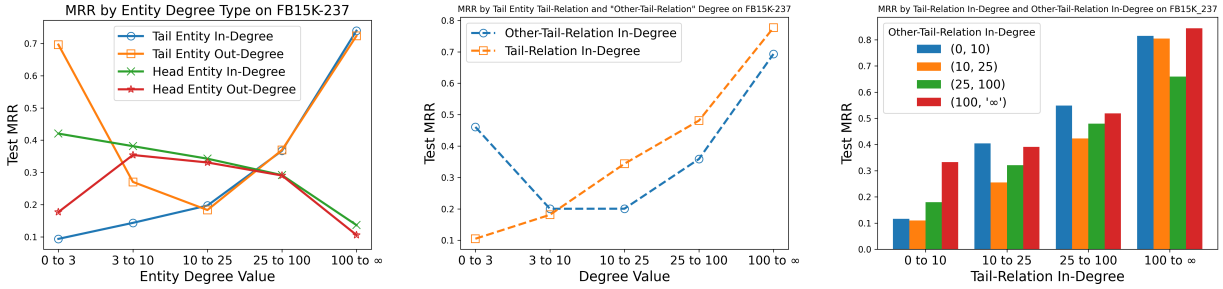
4.2 Related Work

KG Embedding: TransE [13] models the embeddings of a single triple as a translation in the embedding space. Multiple works model the triples as a tensor factorization problem, including [84, 127, 7, 4]. ConvE [29] learns the embeddings by modeling the interaction of a single triple via a convolutional neural network. Other methods like R-GCN [100] modify GCN [55] for relational graphs.

Imbalanced/Long-Tail Learning: Imbalanced/Long-Tail Learning considers the problem of model learning when the class distribution is highly uneven. SMOTE [21], a classic technique, attempts to produce new synthetic samples for the minority class. Recent work has focused on tackling imbalance problems on deeper models. Works such as [90, 106, 66] address this problem by modifying the loss for different samples. Another branch of work tries to tackle this issue by utilizing ensemble modeling [119, 150, 122].

Degree Bias: [76] demonstrate the existence of popularity bias in popular KG datasets, which causes models to inflate the score of entities with a high degree. [11] show the existence of entity degree bias in biomedical KGs. [93] demonstrate that the performance is positively correlated with the number of source peers and negatively with the number of target peers. [56] analyze the degree bias of random walks. To alleviate this issue, they propose a debiasing method that utilizes random graphs. In addition, many studies have focused on allaying the effect of degree bias for the task of node classification including [107, 146, 68]. However, there is no work that focuses on how the intersection of entity and relation degree bias effects embedding-based KGC.

Data Augmentation for Graphs There is a line of works studying data augmentation for homogeneous graphs [149, 147]. Few of these works study the link prediction problem [148] but they



(a) In and Out Degree Analysis (b) Tail In-Degree Analysis (c) Controlling for Degrees

Figure 4.2 MRR when predicting the tail for TuckER on FB15K-237 when varying the (a) in-degree and out-degree of the head and tail entity, (b) tail-relation and other-relation in-degree, and (c) other-relation in-degree for smaller sub-ranges of the tail-relation degree.

do not address the issues in KGs. To augment KGs, [108] generate synthetic triples via adversarial learning; [118] use a GAN to create stronger negative samples; [59] use rule mining to identify new triples to augment the graph with. However, all these methods do not augment with for the purpose of degree bias in KGs and hence are not applicable to the problem this paper studies.

4.3 Preliminary Study

In this section we empirically study the degree bias problem in KGC. We focus on two representative embedding based methods ConvE [29] and TuckER [7].

We first introduce some notations. We denote $\mathcal{G} = \{\mathcal{V}, \mathcal{R}, \mathcal{E}\}$ as a KG with entities $\mathcal{V}$, relations $\mathcal{R}$, and edges $\mathcal{E}$. Each edge represents two entities connected by a single relation. We refer to an edge as a triple and denote it as (h, r, t) where h is referred to as the head entity, t the tail entity, and r the relation. Each entity and relation is represented by an embedding. We represent the embedding for a single entity v as $\mathbf{x}_v \in \mathbb{R}^{n_v}$ and the embedding for a relation r as $\mathbf{x}_r \in \mathbb{R}^{n_r}$, where n_v and n_r are the dimensions of the entity and relation embeddings, respectively. We further define the degree of an entity v as d_v and the in-degree ($d_v^{(in)}$) and out-degree ($d_v^{(out)}$) as the number of triples where v is the tail and head entity, respectively.

Lastly, KGC attempts to predict new facts that are not found in the original KG. This involves predicting the tail entities that satisfy $(h, r, ?)$ and the head entities that satisfy $(?, r, t)$. Following [30], we augment all triples (h, r, t) with its inverse (t, r^{-1}, h). As such, predicting the head

entity of (h, r, t) is analogous to predicting the tail entity for $(t, r^{-1}, ?)$. Under such a setting, KGC can be formulated as always predicting the tail entity. *Therefore, in the remainder of this work, we only consider KGC as predicting the tail entities that satisfy* $(h, r, ?)$.

In the following subsections, we will explore the following questions: (1) Does degree bias exist in typical KG embedding models? and (2) Which factor in a triple is related to such bias? To answer these questions, we first study how the head and tail entity degree affect KGC performance in Section 4.3.1. Then, we investigate the impact of the frequency of entity-relation pairs co-occurring on KGC performance in Section 4.3.2.

4.3.1 Entity Degree Analysis

We first examine the effect that the degree of both the head and tail entities have on KGC performance. We perform our analysis on the FB15K-237 dataset [111], a commonly used benchmark in KGC. Since a KG is a directed graph, we postulate that the direction of an entity's edges matters. We therefore split the degree of each entity into its in-degree and out-degree. We measure the performance using the mean reciprocal rank (MRR). Note that the degree metrics are calculated using only the training set.

Figure 4.2a displays the results of TuckER (see Section D.3.5 for more details) on FB15K-237 split by both entities and degree type. From Figure 4.2a we observe that when varying the tail entity degree value, the resulting change in test MRR is significantly larger than when varying the degree of head entities. Furthermore, the MRR increases drastically with the increase of tail in-degree (blue line) while there is a parabolic-like relationship when varying the tail out-degree (orange line). From these observations we can conclude: (1) the degree of the tail entity (i.e. the entity we are trying to predict) has a larger impact on test performance than the degree of the head entity; (2) the tail in-degree features a more distinguishing and apparent relationship with performance than the tail out-degree. Due to the page limitation, the results of ConvE are shown in Appendix C.4, where we have similar observations. These results suggest that KGC displays a degree bias in regards to the in-degree. Next, we will examine which factors of a triple majorly contribute to such degree bias.

4.3.2 Entity-Relation Degree Analysis

In the previous subsection, we have demonstrated the relationship between the entity degree and KGC performance. However, it doesn't account for the interaction of the entities and relation. Therefore, we further study how the presence of both the relation and entities in a triple *together* impact the KGC performance. We begin by defining the number of edges that contains the relation r and an entity v as the *relation-specific* degree:

$$d_{v,r} = |\{(h, r, t) \in \mathcal{E} \mid h = v \vee t = v\}|. \tag{4.1}$$

Based on the results in Section 4.3.1, we posit that the main indicator of performance is the in-degree of the tail entity. We extend this idea to our definition of relation-specific degree by only counting co-occurrences of an entity and relation when the entity occurs as the tail. For simplicity we refer to this as the *tail-relation* degree and define it as:

$$d_{v,r}^{(tail)} = |\{(h, r, v) \in \mathcal{E}\}|. \tag{4.2}$$

The tail-relation degree can be understood as the number of edges that an entity v shares with r, where v occupies the position we are trying to predict (i.e. the tail entity). We further refer to the number of in-edges that v doesn't share with r as "Other-Tail Relation" degree. This is calculated as the difference between the in-degree of entity v and the tail-relation degree of v and relation r, i.e. $d_v^{(in)} - d_{v,r}^{(tail)}$. It is easy to verify that the in-degree of an entity v is the summation of the tail-relation degree and "Other-Tail Relation" degree. We use Figure 4.1 as an example of the tail-relation degree. The entity *Sweden* co-occurs with the relation *Has Country* on one edge. On that edge, *Sweden* is the tail entity. Therefore the tail-relation degree of the pair (*Sweden*, *Has Country*) is one. We note that a special case of the tail-relation degree is relation-level semantic evidence defined by [64].

Figure 4.2b displays the MRR when varying the value of the tail-relation and "Other-Tail Relation" degree of the tail entity. From the results, we note that while both degree metrics correlate with performance, the performance when the other-tail-relation degree in the range $[0, 3)$ is quite high. Since both metrics are highly correlated, it is difficult to determine which metric

is more important for the downstream performance. Is the "Other-Tail Relation" the determining factor for performance or is it the tail-relation degree? We therefore check the performance when controlling for one another. Figure 4.2c displays the results when varying the "Other-Tail Relation" degree for specific sub-ranges of the tail-relation degree. From this figure, we see that the tail-relation degree exerts a much larger influence on the KGC performance as there is little variation between bars belonging to the same subset. Rather the tail-relation degree (i.e. the clusters of bars) has a much larger impact. Therefore, we conclude that for a single triple, the main factor of degree bias is the tail-relation degree of the tail entity.

Remark. Our analysis differs from traditional research on degree bias. While previous works focus only on the degree of the node, we focus on a specific type of frequency among entity-relation pairs. This is vital as the frequencies of both the entities and relations are important in KGs. Though we only analyze KGs, findings from our analysis could be applicable to other types of heterogeneous graphs.

4.4 The Proposed Method

Grounded in the observations in Section 4.3.2, one natural idea to alleviate the degree bias in KGC is to compensate the triples with low tail-relation degrees. Based on this intuition, we propose a new method for improving the KGC performance of triples with low tail-relation degrees. Our method, **KG-Mixup**, works by augmenting the low tail-relation degree triples during training with synthetic samples. This strategy has the effect of increasing the degree of an entity-relation pair with a low tail-relation degree by creating more shared edges between them. Therefore, KG-Mixup is very general and can further be used in conjunction with any KG embedding technique.

4.4.1 General Problem

In Section 4.3.2 we showed that the tail-relation degree of the tail entity strongly correlates with higher performance in KGC. As such we seek to design a method that can increase the performance of such low-degree entity-relation pairs without sacrificing the performance of high-degree pairs.

To solve this problem, we consider data augmentation. Specifically, we seek to create synthetic triples for those entity-relations pairs with a low tail-relation degree. In such a way we are creating

more training triples that contain those pairs, thereby "increasing" their degree. For each entity-relation pair with a tail-relation degree less than η, we add k synthetic samples, which can be formulated as follows:

$$\tilde{\mathcal{E}}_{v,r} = \begin{cases} \mathcal{E}_{v,r} \cup \{(\tilde{h}, \tilde{r}, \tilde{t})\}_{i=1}^{k} & d_{v,r}^{(tail)} < \eta, \\ \mathcal{E}_{v,r} & \text{else}, \end{cases} \tag{4.3}$$

where $(h, r, v) \in \mathcal{E}_{v,r}$ are the original training triples with the relation r and the tail entity v, $(\tilde{h}, \tilde{r}, \tilde{t})$ is a synthetic sample, and $\tilde{\mathcal{E}}_{v,r}$ is the new set of triples to use during training.

Challenges. We note that creating the synthetic samples as shown in Eq. (4.3) is non-trivial and there are a number of challenges:

1. How do we produce the synthetic samples for KG triples that contain multiple types of embeddings?
2. How do we promote diversity in the synthetic samples $(\tilde{h}, \tilde{r}, \tilde{t})$? We want them to contain sufficient information from the original entity and relation embeddings we are augmenting, while also being distinct from similar triples in $\mathcal{E}_{v,r}$.
3. How do we achieve such augmentation in a computationally efficient manner?

These challenges motivate us to design a special data augmentation algorithm for knowledge graph completion and we detail its core techniques in the next subsection.

4.4.2 KG-Mixup

We now present our solution for producing synthetic samples as described in Eq. (4.3). Inspired by the popular Mixup [137] strategy, we strive to augment the training set by mixing the representations of triples. We draw inspiration from mixup as (1) it is an intuitive and widely used data augmentation method, (2) it is able to promote diversity in the synthetic samples via the randomly drawn value λ, and (3) it is computationally efficient (see Section D.2.2).

We now briefly describe the general mixup algorithm. We denote the representations of two samples as x_1 and x_2 and their labels y_1 and y_2. Mixup creates a new sample $\tilde{x}$ and label $\tilde{y}$ by combining both the representations and labels via a random value $\lambda \in [0, 1]$ drawn from

Algorithm 4.1 KG-Mixup Training Procedure

Require:
$G = \{V, R, \mathcal{E}\}$ ▷ Training graph
k, η ▷ # of samples to generate and degree threshold
X, W ▷ Model embeddings and parameters

Randomly initialize X and W
Pre-train to obtain initial X
Randomly re-initialize W
while not converged **do**
  for $e = (h_i, r_i, t_i) \in \mathcal{E}$ **do**
    if $d^{(tail)}_{t_i, r_i} < \eta$ **then**
      $C = \{(h^*, r^*, t_i) \in \mathcal{E}\}$
      S = Rand-Sample(C, k)
      $\mathcal{E}_{\text{mix}} = \{\text{Mix}(e, s)\ \forall s \in S\}$ ▷ Eq. (4.6)
    else
      $\mathcal{E}_{\text{mix}} = \{\}$
    end if
    Update model parameters on $\{e\} \cup \mathcal{E}_{\text{mix}}$
  end for
end while
return X and W

$\lambda \sim \text{Beta}(\alpha, \alpha)$ such that:

$$\tilde{x} = \lambda \mathbf{x}_1 + (1 - \lambda)\mathbf{x}_2, \tag{4.4}$$

$$\tilde{y} = \lambda \mathbf{y}_1 + (1 - \lambda)\mathbf{y}_2. \tag{4.5}$$

We adapt this strategy to our studied problem for a triple (h, r, t) where the tail-relation degree is below a degree threshold, i.e. $d^{(tail)}_{t,r} < \eta$. For such a triple we aim to augment the training set by creating k synthetic samples $\{(\tilde{h}, \tilde{r}, \tilde{t})\}_{i=1}^{k}$. This is done by mixing the original triple with k other triples $\{(h_i, r_i, t_i)\}_{i=1}^{k}$.

However, directly adopting mixup to KGC leads to some problems: (1) Since each sample doesn't contain a label (Eq. 4.5) we are unable to perform label mixing. (2) While standard mixup randomly selects samples to mix with, we may want to utilize a different selection criteria to better enhance those samples with a low tail-relation degree. (3) Since each sample is composed of multiple components (entities and relations) it's unclear how to mix two samples. We go over these challenges next.

4.4.2.1 Label Incorporation in KGC

We first tackle how to incorporate the label information as shown in Eq. (4.5). Mixup was originally designed for classification problems, making the original label mixing straightforward. However, for KGC, we have no associated label for each triple. We therefore consider the entity we are predicting as the label. For a triple $e_1 = (h_1, r_1, t_1)$ where we are predicting t_1, the label would be considered the entity t_1.

4.4.2.2 Mixing Criteria

Per the original definition of Mixup, we would then mix e_1 with a triple belonging to the set $\{(h_2, r_2, t_2) \in \mathcal{E} \mid t_2 \neq t_1\}$. However, since our goal is to predict t_1 we wish to avoid mixing it. Since we want to better predict t_1, we need to preserve as much tail (i.e. label) information as possible. As such, we only consider mixing with other triples that share the same tail and belong to the set $\{(h_2, r_2, t_1) \in \mathcal{E} \mid h_1 \neq h_2, r_1 \neq r_2\}$. Our design is similar to SMOTE [21], where only samples belonging to the same class are combined. We note that while it would be enticing to only consider mixing with triples containing the same entity-relation pairs, i.e. $(h_2, r_1, t_1) \in \mathcal{E}_{t_1,r_1}$, this would severely limit the number of possible candidate triples as the tail-relation degree can often be as low as one or two for some pairs.

4.4.2.3 How to Mix?

We now discuss how to perform the mixing of two samples. Given a triple $e_1 = (h_1, r_1, t_1)$ of low tail-relation degree we mix it with another triple that shares the same tail (i.e. label) such that $e_2 = (h_2, r_2, t_1)$. Applying Eq. (4.4) to e_1 and e_2, a synthetic triple $\tilde{e} = (\tilde{h}, \tilde{r}, \tilde{t})$ is equal to:

$$\tilde{e} = \lambda e_1 + (1 - \lambda) e_2, \tag{4.6}$$

$$\tilde{e} = \lambda (h_1, r_1, t_1) + (1 - \lambda)(h_2, r_2, t_1), \tag{4.7}$$

$$\tilde{e} = \lambda (\mathbf{x}_{h_1}, \mathbf{x}_{r_1}, \mathbf{x}_{t_1}) + (1 - \lambda)(\mathbf{x}_{h_2}, \mathbf{x}_{r_2}, \mathbf{x}_{t_1}), \tag{4.8}$$

where $\mathbf{x}_{h_i}$ and $\mathbf{x}_{r_j}$ represent the entity and relation embeddings, respectively. We apply the weighted sum to the head, relation, and tail, separately. Each entity and relation are therefore equal to:

$$x_{\tilde{h}} = \lambda \mathbf{x}_{h_1} + (1-\lambda)\mathbf{x}_{h_2}, \tag{4.9}$$

$$x_{\tilde{r}} = \lambda \mathbf{x}_{r_1} + (1-\lambda)\mathbf{x}_{r_2}, \tag{4.10}$$

$$x_{\tilde{t}} = \mathbf{x}_{t_1}. \tag{4.11}$$

We use Figure 4.1 to illustrate an example. Let e_1 = (*Europe*, *Has Country*, *Germany*) be the triple we are augmenting. We mix it with another triple with the tail entity *Germany*. We consider the triple e_2 = (*Belgium*, *Borders*, *Germany*). The mixed triple is represented as $\tilde{e}$ = (*Europe*+*Belgium*, *Has Country*+*Borders*, *Germany*). As e_1 contains the continent that *Germany* belongs to and e_2 has the country it borders, we can understand the synthetic triple $\tilde{e}$ as conveying the geographic location of *Germany* inside of *Europe*. This is helpful when predicting *Germany* in the original triple e_1, since the synthetic sample imbues the representation of *Germany* with more specific geographic information.

4.4.3 KG-Mixup Algorithm for KGC

We utilize the binary cross-entropy loss when training each model. The loss is optimized using the Adam optimizer [52]. We also include a hyperparameter β for weighting the loss on the synthetic samples. The loss on a model with parameters θ is therefore:

$$\mathcal{L}(\theta) = \mathcal{L}_{KG}(\theta) + \beta \mathcal{L}_{\text{Mix}}(\theta)\,, \tag{4.12}$$

where $\mathcal{L}_{KG}$ is the loss on the original KG triples and $\mathcal{L}_{\text{Mix}}$ is the loss on the synthetic samples. The full algorithm is displayed in Algorithm 4.1. We note that we first pre-train the model before training with KG-Mixup, to obtain the initial entity and relation representations. This is done as it allows us to begin training with stronger entity and relation representations, thereby improving the generated synthetic samples.

4.4.4 Algorithmic Complexity

We denote the algorithmic complexity of a model f (e.g. ConvE [29] or TuckER [7]) for a single sample e as $O(f)$. Assuming we generate N negative samples per training sample, the

training complexity of f over a single epoch is:

$$O\left(N \cdot |\mathcal{E}| \cdot O(f)\right), \tag{4.13}$$

where $|\mathcal{E}|$ is the number of training samples. In KG-Mixup, in addition to scoring both the positive and negative samples, we also score the synthetic samples created for all samples with a tail-relation degree below a threshold η. We refer to that set of samples below the degree threshold as $\mathcal{E}_{\text{thresh}}$. We create k synthetic samples per $e \in \mathcal{E}_{\text{thresh}}$. As such, our algorithm scores an additional $k \cdot |\mathcal{E}_{\text{thresh}}|$ samples for a total of $N \cdot |\mathcal{E}| + k \cdot |\mathcal{E}_{\text{thresh}}|$ samples per epoch. Typically the number of negative samples $N >> k$. Both ConvE and TuckER use all possible negative samples per training sample while we find $k = 5$ works well. Furthermore, by definition, $\mathcal{E}_{\text{thresh}} \subseteq \mathcal{E}$ rendering $|\mathcal{E}| >> |\mathcal{E}_{\text{thresh}}|$. We can thus conclude that $N \cdot |\mathcal{E}| >> k \cdot |\mathcal{E}_{\text{thresh}}|$. We can therefore express the complexity of KG-Mixup as:

$$\approx O\left(N \cdot |\mathcal{E}| \cdot O(f)\right). \tag{4.14}$$

This highlights the efficiency of our algorithm as its complexity is approximately equivalent to the standard training procedure.

4.5 Regularizing Effect of KG-Mixup

In this section, we examine the properties of KG-Mixup and show it can be formulated as a form of regularization on the entity and relation embeddings of low tail-relation degree samples following previous works [19, 138].

We denote the mixup loss with model parameters θ over samples S as $\mathcal{L}_{\text{Mix}}(\theta)$. The set S contains those samples with a tail-relation degree below a threshold η (see line 6 in Algorithm 4.1). The embeddings for each sample $e_i = (h_i, r_i, t) \in S$ is mixed with those of a random sample $e_j = (h_j, r_j, t)$ that shares the same tail. The embeddings are combined via a random value $\lambda \sim \text{Beta}(\alpha, \alpha)$ as shown in Eq. (4.9), thereby producing the synthetic sample $\tilde{e} = (\tilde{h}, \tilde{r}, t)$. The formulation for $\mathcal{L}_{\text{mix}}(\theta)$ is therefore:

$$\mathcal{L}_{\text{Mix}}(\theta) = \frac{1}{k|S|} \sum_{i=1}^{|S|} \sum_{j=1}^{k} l_\theta\left(\tilde{e}, \tilde{y}\right), \tag{4.15}$$

where k synthetic samples are produced for each sample in S, and $\tilde{y}$ is the mixed binary label. Following Theorem 1 in [19] we can rewrite the loss as the expectation over the synthetic samples as,

$$\mathcal{L}_{\text{Mix}}(\theta) = \frac{1}{|S|} \sum_{i=1}^{|S|} \mathbb{E}_{\lambda,j}\, l_\theta\left(\tilde{e}, \tilde{y}\right), \tag{4.16}$$

where $\lambda \sim \mathcal{D}_\lambda$ and $j \sim \text{Uniform}(\mathcal{E}_t)$. The distribution $\mathcal{D}_\lambda = \text{Beta}_{[\frac{1}{2},1]}(\alpha, \alpha)$ and the set $\mathcal{E}_t$ contains all samples (h_j, r_j, t) with tail t. Since the label y for both samples i and j are always 1, rendering $\tilde{y} = 1$, we can simplify Eq. (4.16) arriving at:

$$\mathcal{L}_{\text{Mix}}(\theta) = \frac{1}{|S|} \sum_{i=1}^{|S|} \mathbb{E}_{\lambda,j}\, l_\theta\left(\tilde{e}\right). \tag{4.17}$$

For the above loss function, we have the following theorem.

Theorem 1. *The mixup loss $\mathcal{L}_{Mix}(\theta)$ defined in Eq.* (4.17) *can be rewritten as the following where the loss function l_θ is the binary cross-entropy loss, $\mathcal{L}(\theta)$ is the loss on the original set of augmented samples S, and $\mathcal{R}_1(\theta)$ and $\mathcal{R}_2(\theta)$ are two regularization terms,*

$$\mathcal{L}_{Mix}(\theta) = \mathcal{L}(\theta) + \mathcal{R}_1(\theta) + \mathcal{R}_2(\theta). \tag{4.18}$$

The regularization terms are given by the following where each mixed sample $\tilde{e}$ is composed of a low tail-relation degree sample e_i and another sample with the same tail entity e_j:

$$\mathcal{R}_1(\theta) = \frac{\tau}{|S|} \sum_{i=1}^{|S|} \sum_{j=1}^{k} \left(1 - \sigma\left(f(e_i)\right)\right) \frac{\partial f(e_i)^T}{\partial x_{\tilde{h}}} \Delta h, \tag{4.19}$$

$$\mathcal{R}_2(\theta) = \frac{\tau}{|S|} \sum_{i=1}^{|S|} \sum_{j=1}^{k} \left(1 - \sigma\left(f(e_i)\right)\right) \frac{\partial f(e_i)^T}{\partial x_{\tilde{r}}} \Delta r, \tag{4.20}$$

with $\tau = \mathbb{E}_{\lambda\sim\mathcal{D}_\lambda}(1-\lambda)$, $\Delta h = \left(x_{h_j} - x_{h_i}\right)$, $\Delta r = \left(x_{r_j} - x_{r_i}\right)$, σ is the sigmoid function, and f is the score function.

We provide the detailed proof of Theorem 1 in Appendix C.6. Examining the terms in Eq (4.18), we can draw the following understandings on KG-Mixup:

1. The inclusion of $\mathcal{L}(\theta)$ implies that the low tail-relation degree samples are scored an additional time when being mixed. This can be considered as a form of oversampling on the low tail-relation degree samples.
2. If the probability is very high, i.e. $\sigma(f(e_i)) \approx 1$, both $\mathcal{R}_1$ and $\mathcal{R}_2$ cancel out. This is intuitive as if the current parameters perform well for the original low-degree sample, there is no need to make any adjustments.
3. We can observe that $\mathcal{R}_1$ and $\mathcal{R}_2$ enforce some regularization on the derivatives as well as the difference between the embeddings Δh and Δr. This motivates us to further examine the difference between the embeddings. In Section 4.6.3, we find that our method does indeed produce more similar embeddings, indicating that our method exerts a smoothing effect among mixed samples.

4.6 Experiment

In this section we conduct experiments to demonstrate the effectiveness of our approach on multiple benchmark datasets. We further compare the results of our framework to other methods commonly used to address bias. In particular we study if KG-Mixup can (a) improve overall KGC performance and (b) increase performance on low tail-relation degree triples without degrading performance on other triples. We further conduct studies examining the effect of the regularization terms, ascertaining the importance of each component in our framework, and the ability of KG-Mixup to improve model calibration.

4.6.1 Experimental Settings

4.6.1.1 Datasets

We conduct experiments on three datasets including FB15K-237 [111], CoDEx-M [97], and NELL-995 [126]. We omit the commonly used dataset WN18RR [29] as a majority of entities have a degree less than or equal to 3, and as such does not exhibit any degree bias towards triples with a low tail-relation degree. The statistics of each dataset is shown in Table C.1.

4.6.1.2 Baselines

We compare the results of our method, KG-Mixup, with multiple popular methods proposed for addressing imbalanced problems. Such methods can be used to mitigate bias caused by the initial imbalance. In our case, an imbalance in tail-relation degree causes algorithms to be biased against triples of low tail-relation degree. Specifically, we implement: (a) **Over-Sampling** triples below a degree threshold η. We over-sample $\eta - d_{v,r}^{(tail)}$ times, (b) **Loss Re-Weighting** [135], which assigns a higher loss to triples with a low tail-relation degree, (c) **Focal Loss** [66], which assigns a higher weight to misclassified samples (e.g. low degree triples).

4.6.1.3 Evaluation Metrics

To evaluate the model performance on the test set, we report the mean reciprocal rank (MRR) and the Hits@k for $k = 1, 10$. Following [13], we report the performance using the filtered setting.

4.6.1.4 Implementation Details

In this section, we detail the training procedure used to train our framework KG-Mixup. We conduct experiments on our framework using two different KG embedding models, ConvE [29] and TuckER [7]. Both methods are widely used to learn KG embeddings and serve as a strong indicator of our framework's efficacy. We use stochastic weight averaging (SWA) [48] when training our model. SWA uses a weighted average of the parameters at different checkpoints during training for inference. Previous work [89] has shown that SWA in conjunction with data augmentation can increase performance. Lastly, the synthetic loss weighting parameter β is determined via hyperparameter tuning on the validation set.

4.6.2 Main Results

In this subsection we evaluate KG-Mixup on multiple benchmarks, comparing its test performance against the baseline methods. We first report the overall performance of each method on the three datasets. We then report the performance for various degree bins. The top results are bolded with the second best underlined. Note that the **Standard** method refers to training without any additional method to alleviate bias.

Table 4.1 contains the overall results on each method and dataset. The performance is reported

Table 4.1 Knowledge Graph Completion (KGC) Comparison.

Model	Method	FB15K-237			NELL-995			CoDEx-M		
		MRR	H@1	H@10	MRR	H@1	H@10	MRR	H@1	H@10
ConvE	Standard	33.04	23.95	51.23	50.87	**44.14**	61.48	31.70	**24.34**	45.60
	+ Over-Sampling	30.45	21.85	47.81	48.63	40.99	60.78	27.13	20.17	40.11
	+ Loss Re-weighting	32.32	23.32	50.19	50.89	43.83	62.17	28.38	21.12	42.68
	+ Focal Loss	32.08	23.29	50.09	50.43	44.00	60.70	27.99	20.93	41.48
	+ KG-Mixup (Ours)	**34.33**	**25.00**	**53.11**	**51.08**	43.52	**63.22**	**31.71**	23.49	**47.49**
TuckER	Standard	35.19	26.06	53.47	52.11	45.51	**62.26**	31.67	**24.46**	45.73
	+ Over-Sampling	34.77	25.48	53.53	50.36	44.04	60.40	29.97	22.27	44.19
	+ Loss Re-weighting	35.25	26.08	53.34	51.91	45.76	61.05	31.58	24.32	45.41
	+ Focal Loss	34.02	24.79	52.48	49.57	43.28	58.91	31.47	24.05	45.60
	+ KG-Mixup (Ours)	**35.83**	**26.37**	**54.78**	**52.24**	**45.78**	62.14	**31.90**	24.15	**46.54**

Table 4.2 MRR for tail-relation degree bins. The range for the zero, low, medium and high bins are [0, 1), [1, 10), [10, 50), and [50, ∞), respectively.

Model	Method	FB15K-237				NELL-995				CoDEx-M			
		Zero	Low	Medium	High	Zero	Low	Medium	High	Zero	Low	Medium	High
ConvE	Standard	7.34	12.35	34.95	**70.97**	35.37	57.16	**65.99**	**91.90**	8.38	7.97	**34.64**	**65.29**
	+ Over-Sampling	8.37	12.45	33.01	68.75	**36.67**	57.33	56.09	79.57	8.09	7.52	29.51	54.80
	+ Loss Re-weighting	5.03	9.89	30.56	63.34	36.16	57.96	63.69	89.52	8.79	7.09	29.09	58.10
	+ Focal Loss	7.52	11.89	33.96	68.75	34.72	58.00	65.60	90.89	6.78	6.80	33.42	56.96
	+ KG-Mixup (Ours)	**10.90**	**13.92**	**35.74**	70.72	35.38	**59.56**	65.41	90.64	**9.74**	**8.96**	32.63	64.38
TuckER	Standard	10.41	14.65	38.49	71.39	**37.02**	58.21	69.17	90.55	9.99	8.29	**35.23**	63.94
	+ Over-Sampling	**12.25**	14.28	36.79	70.50	34.50	55.46	65.68	**93.47**	**10.98**	7.76	32.50	60.25
	+ Loss Re-weighting	10.61	14.40	37.66	**72.28**	36.59	59.00	67.19	91.17	10.44	8.62	35.00	63.39
	+ Focal Loss	10.84	13.53	37.00	69.28	34.18	53.60	62.67	91.02	9.68	8.17	33.95	64.13
	+ KG-Mixup (Ours)	11.83	**15.61**	**39.45**	70.86	36.12	**60.73**	**71.67**	92.27	9.14	**8.70**	32.38	**65.28**

for both ConvE and TuckER. KG-Mixup achieves for the best MRR and Hits@10 on each dataset for ConvE. For TuckER, KG-Mixup further achieves the best MRR on each dataset and the top Hits@10 for two. Note that the other three baseline methods used for alleviating bias, on average, perform poorly. This may be due to their incompatibility with relational structured data where each sample contains multiple components. It suggests that we need dedicated efforts to handle the degree bias in KGC.

We further report the MRR of each method for triples of different tail-relation degree. We split the triples into four degree bins of zero, low, medium and high degree. The range of each bin is [0, 1), [1, 10], [10, 50), and [50, ∞), respectively. KG-Mixup achieves a notable increase in

performance on low tail-relation degree triples for each dataset and embedding model. KG-Mixup increases the MRR on low degree triples by 9.8% and 5.3% for ConvE and TuckER, respectively, over the standard trained models on the three datasets. In addition to the strong increase in low degree performance, KG-Mixup is also able to retain its performance for high degree triples. The MRR on high tail-relation degree triples degrades, on average, only 1% on ConvE between our method and standard training and actually increases 1% for TuckER. Interestingly, the performance of KG-Mixup on the triples with zero tail-relation degree isn't as strong as the low degree triples. We argue that such triples are more akin to the zero-shot learning setting and therefore different from the problem we are studying.

Lastly, we further analyzed the improvement of KG-Mixup over standard training by comparing the difference in performance between the two groups via the paired t-test. We found that for the results in Table 4.1, $5/6$ are statistically significant ($p<0.05$). Furthermore, for the performance on low tail-relation degree triples in Table 4.2, all results ($6/6$) are statistically significant. This gives further justification that our method can improve both overall and low tail-relation degree performance.

4.6.3 Regularization Analysis

In this subsection we empirically investigate the regularization effects of KG-Mixup discussed in Section 4.5. In Section 4.5 we demonstrated that KG-Mixup can be formulated as a form of regularization. We further showed that one of the quantities minimized is the difference between the head and relation embeddings of the two samples being mixed, e_i and e_j, such that $(x_{h_j} - x_{h_i})$ and $(x_{r_j} - x_{r_i})$. Here e_i is the low tail-relation degree sample being augmented and e_j is another sample that shares the same tail. We deduce from this that for low tail-relation degree samples, KG-Mixup may cause their head and relation embeddings to be more similar to those of other samples that share same tail. Such a property forms a smoothing effect on the mixed samples, which facilitates a transfer of information to the embeddings of the low tail-relation degree sample.

We investigate this by comparing the head and relation embeddings of all samples that are augmented with all the head and relation embeddings that also share the same tail entity. We

denote the set of all samples below some tail-relation degree threshold η as $\mathcal{E}_{\text{thresh}}$ and all samples with tail entity t as $\mathcal{E}_t$. Furthermore, we refer to all head entities that are connected to a tail t as $\mathcal{H}_t = \{h_j \mid (h_j, r_j, t) \in \mathcal{E}_t\}$ and all such relations as $\mathcal{R}_t = \{r_j \mid (h_j, r_j, t) \in \mathcal{E}_t\}$. For each sample $(h_i, r_i, t) \in \mathcal{E}_{\text{thresh}}$ we compute the mean euclidean distance between the (1) head embedding $\mathbf{x}_{h_i}$ and all $\mathbf{x}_{h_j} \in \mathcal{H}_t$ and (2) the relation embedding $\mathbf{x}_{r_i}$ and all $\mathbf{x}_{r_j} \in \mathcal{R}_t$. For a single sample e_i the mean head and relation embedding distance are given by $h_{\text{dist}}(e_i)$ and $r_{\text{dist}}(e_i)$, respectively. Lastly, we take the mean of both the head and relation embeddings mean distances across all $e \in \mathcal{E}_{\text{thresh}}$,

$$D_{\text{rel}} = \text{Mean}\left(r_{\text{dist}}(e_i) \mid e_i \in \mathcal{E}_{\text{thresh}}\right), \tag{4.21}$$

$$D_{\text{head}} = \text{Mean}\left(h_{\text{dist}}(e_i) \mid e_i \in \mathcal{E}_{\text{thresh}}\right). \tag{4.22}$$

Both D_{head} and D_{rel} are shown in Table 4.3 for models fitted with and without KG-Mixup. We display the results for ConvE on FB15K-237. For both the mean head and relation distances, KG-Mixup produces smaller distances than the standardly-trained model. This aligns with our previous theoretical understanding of the regularization effect of the proposed method: for samples for which we augment during training, their head and relation embeddings are more similar to those embeddings belonging to other samples that share the same tail. This to some extent forms a smoothing effect, which is helpful for learning better representations for the low-degree triplets.

Table 4.3 Mean Embedding Distances on FB15K-237.

Embedding Type	**Head Entity**	**Relation**
w/o KG-Mixup	1.18	1.21
KG-Mixup	1.09	1.13
% Decrease	-7.6%	-6.6%

4.6.4 Ablation Study

In this subsection we conduct an ablation study of our method on the FB15K-237 dataset using ConvE and TuckER. We ablate both the data augmentation strategy and the use of stochastic weight averaging (SWA) separately to ascertain their effect on performance. We report the overall test MRR and the low tail-relation degree MRR. The results of the study are shown in Table 4.4.

KG-Mixup achieves the best overall performance on both embedding models. Using only our data augmentation strategy leads to an increase in both the low degree and overall performance. On the other hand, while the SWA-only model leads to an increase in overall performance it degrades the low degree performance. We conclude from these observations that data augmentation component of KG-Mixup is vital for improving low degree performance while SWA helps better maintain or even improve performance on the non-low degree triples.

Table 4.4 Ablation Study on FB15K-237.

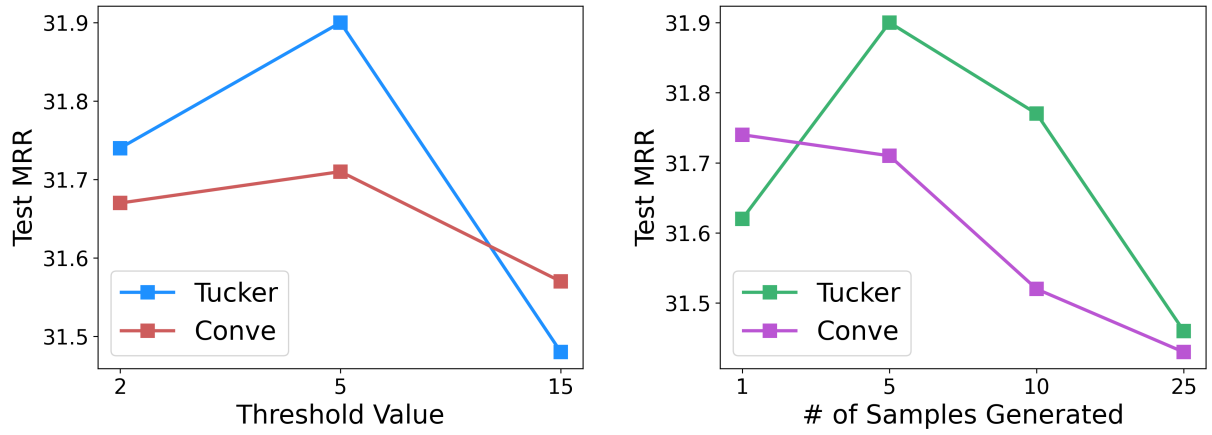
Method	**ConvE**		**TuckER**	
	Low	Overall	Low	Overall
Standard	12.35	33.04	14.65	35.19
+ SWA	12.27	33.69	14.18	35.77
+ Augmentation	**13.99**	33.67	**15.64**	35.62
KG-Mixup (Ours)	13.92	**34.33**	15.61	**35.83**

4.6.5 Parameter Study

In this subsection we study how varying the number of generated synthetic samples k and the degree threshold η affect the performance of KG-Mixup. We consider the values $k \in \{1, 5, 10, 25\}$ and $\eta \in \{2, 5, 15\}$. We report the MRR for both TuckER and ConvE on the CoDEx-M dataset. Figure 4.3a displays the performance when varying the degree threshold. Both methods peak at a value of $\eta = 5$ and perform worst at $\eta = 15$. Figure 4.3b reports the MRR when varying the number of synthetic samples generated. Both methods peak early with ConvE actually performing best at $k = 1$. Furthermore, generating too many samples harms performance as evidenced by the sharp drop in MRR occurring after $k = 5$.

4.6.6 Model Calibration

In this subsection we demonstrate that KG-Mixup is effective at improving the calibration of KG embedding models. Model calibration [40] is concerned with how well calibrated a models prediction probabilities are with its accuracy. Previous work [87] have discussed the desirability of calibration to minimize bias between different groups in the data (e.g. samples of differing degree). Other work [116] has drawn the connection between out-of-distribution generalization and model



(a) Varying degree threshold

(b) Varying # of samples

Figure 4.3 MRR of TuckER and ConvE on CoDEx-M (a) when varying the degree threshold and (b) when varying the number of samples generated.

calibration, which while not directly applicable to our problem is still desirable. Relevant to our problem, [110] has shown that Mixup is effective at calibrating deep models for the tasks of image and text classification. As such, we investigate if KG-Mixup is helpful at calibrating KG embedding models for KGC.

Table 4.5 Expected Calibration Error (ECE). Lower is better.

Model	**Method**	**FB15K-237**		**NELL-995**		**CoDEx-M**	
		Low	Overall	Low	Overall	Low	Overall
ConvE	Standard	0.19	0.15	0.34	0.27	0.28	0.26
	KG-Mixup	0.08	0.05	0.08	0.08	0.02	0.09
TuckER	Standard	0.20	0.35	0.63	0.56	0.05	0.34
	KG-Mixup	0.07	0.1	0.26	0.20	0.01	0.06

We compared the expected calibration error (see Appendix C.5 for more details) between models trained with KG-Mixup and those without on multiple datasets. We report the calibration in Table 4.5 for all samples and those with a low tail-relation degree. We find that in every instance KG-Mixup produces a better calibrated model for both ConvE and TuckER. These results suggest another reason for why KG-Mixup works; a well-calibrated model better minimizes the bias between different groups in the data [87]. This is integral for our problem where certain groups

of data (i.e. triples with low tail-relation degree) feature bias.

4.7 Conclusion

We explore the problem of degree bias in KG embeddings. Through empirical analysis we find that when predicting the tail t for a triple (h, r, t), a strong indicator performance is the number of edges where r and t co-occur as the relation and tail, respectively. We refer to this as the tail-relation degree. We therefore propose a new method, KG-Mixup, that can be used in conjunction with any KG embedding technique to improve performance on triples with a low tail-relation degree. It works by augmenting lower degree entity-relation pairs with additional synthetic triples during training. To create synthetic samples we adapt the Mixup [137] strategy to KGs. Experiments validate its usefulness. For future work we plan on expanding our method to path-based techniques such as NBFNet [153].

CHAPTER 5

DISTANCE-BASED PROPAGATION FOR EFFICIENT KNOWLEDGE GRAPH REASONING

5.1 Introduction

Knowledge graphs (KGs) encode facts via edges in a graph. Because of this, one can view the task of predicting unknown edges (i.e. link prediction) as analogous to uncovering new facts. This task is referred to as knowledge graph completion (KGC) and has attracted a bevy of research over the past decade [14, 112, 100, 154]. Most work has focused on learning quality representations for all nodes (i.e. entities) and edge types (i.e. relations) in the graph to facilitate KGC.

Recently, methods [154, 96, 143], have been introduced that move away from the embedding-based approach and focus instead on learning directly from path-based information. One recent GNN-based method, NBFNet [154], draws inspiration from the Bellman-Ford algorithm by computing path information through dynamic programming. By doing so, it learns pairwise embeddings between all node pairs in an inductive fashion. It achieves state-of-the-art performance in both the transductive and inductive KGC settings. In this work, we refer to such methods as path-based GNNs. However, a downside of path-based GNNs is their inefficiency. This limits their ability in large real-world graphs. Furthermore, it inhibits their ability to propagate deeply in the graph. Two recent methods have been proposed to address the inefficiency problem, i.e., A*Net [152] and AdaProp [144], by only propagating to a subset of nodes every iteration. However, they still tend to propagate unnecessary and redundant messages.

For path-based GNNs, only the source node is initialized with a non-zero message at the beginning of the propagation process. Such models often run a total of T layers, where, in each layer, all nodes aggregate messages from their neighboring edges. We identify that this design is inefficient by making the following two observations. (1) **Empty Messages:** In the propagation process, a node only obtains non-empty messages when the number of propagation layers is $\geq$ the shortest path distance between the source and the node. This means that a large number of nodes far from the source node only aggregate "empty" messages in the early propagation

layers. Nonetheless, path-based GNN models such as NBFnet propagate these unnecessary "empty messages" in these early propagation layers. (2) **Redundant Messages:** To ensure path information from the source reach distant nodes, the number of layers T needs to be sufficiently large. However, a large T induces the propagation of redundant messages for those nodes that are close to the source node. Intuitively, short paths contain more significant information than long ones [50]. The "close" nodes typically aggregate enough information from shorter paths in the early propagation layers. Propagating messages for longer paths in later layers for "close" nodes does not provide significant information and needlessly adds to the complexity. More details on these two observations are provided in Section 5.3.1.

To address these limitations and make the propagation process more efficient, we aim to develop an algorithm that limits the propagation of "empty" and "redundant" messages. In particular, we propose a new method **TAGNet** - **T**runc**A**ted propa**G**ation **Net**work. TAGNet only aggregates paths in a fixed window for each source-target pair, which can be considered a form of path pruning. Our contributions can be summarized as follows:

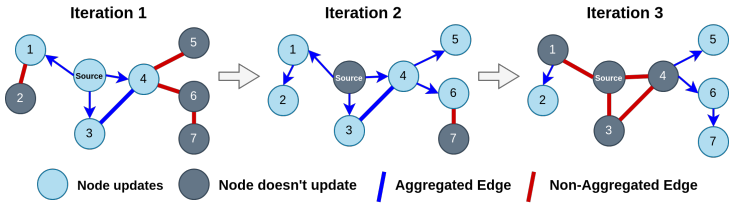
- We propose a new path-based GNN, TAGNet, which customizes the amount of path-pruning for each source-target node pair.
- We demonstrate that the complexity of TAGNet is independent of the number of layers, allowing for efficient deep propagation.
- Extensive experiments demonstrate that TAGNet reduces the number of aggregated messages by up to 90% while matching or even slightly outperforming NBFNet on multiple KG benchmarks.

5.2 Preliminary

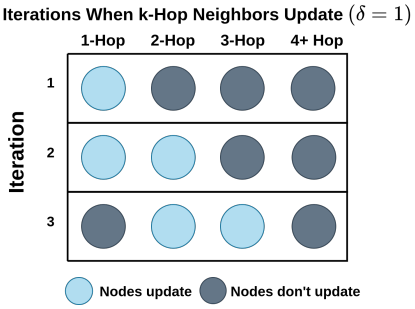
In this section, we first introduce the notation used throughout the paper. We then introduce the path formulation from [154], the generalized Bellman-Ford algorithm [9], and NBFNet [154].

5.2.1 Notations

We denote a KG as $\mathcal{G} = \{\mathcal{V}, \mathcal{R}, \mathcal{E}\}$ with entities $\mathcal{V}$, relations $\mathcal{R}$, and edges $\mathcal{E}$. An edge is denoted as a triple and is of the form (s, q, o) where s is the subject, q the query relation, and o the object. for an incomplete fact $(s, q, ?)$. In such a problem, we refer to the node entity s as



(a) Example of propagating for three iterations with δ=1.



(b) Update status of nodes when δ=1.

Figure 5.1 Example of our algorithm when $\delta = 1$. We note that an **undirected** blue edge indicates that both nodes aggregate each other. A **directed** edge indicates that only the head node aggregates the tail node. E.g., at iteration 2 node 2 aggregates node 1, however node 1 doesn't aggregate node 2.

the source node and any possible answer ? as the target node. Lastly, we denote the shortest path distance between nodes s and o as $\text{dist}(s, o)$. We assume an edge weight of 1 since KGs typically don't contain edge weights.

5.2.2 Path Formulation

[154] introduce a general path formulation for determining the existence of an edge (s, q, o). They consider doing so by aggregating all paths between s and o, conditional on the query q. We denote the maximum path length as T (in their paper they set $T = \infty$), $P^t_{s,o}$ represents all paths of length t connecting nodes s and o, and $\mathbf{w}_q(e_i)$ is the representation of an edge e_i conditional on the relation q. The representation of an edge (s, q, o) is given by $\mathbf{h}_q(s, o)$:

$$\mathbf{h}_q(s,o) = \bigoplus_{t=1}^{T} \bigoplus_{p \in P^t_{s,o}} \bigotimes_{i=1}^{|p|} \mathbf{w}_q(e_i). \tag{5.1}$$

[154] show that this formulation can capture many existing graph algorithms including the Katz index [50], Personalized PageRank [15] and others.

5.2.3 Generalized Bellman-Ford

Due to the exponential relationship between path length and the number of paths, calculating Eq. (5.1) for large T is unfeasible. As such, [154] instead model Eq. (5.1) via the generalized Bellman-Ford algorithm [9] which recursively computes such path information in a more efficient

manner. It is formulated as:

$$\mathbf{h}_q^{(0)}(s,o) = \mathbf{1}_q(s=o), \tag{5.2}$$

$$\mathbf{h}_q^{(t)}(s,o) = \left(\bigoplus_{(x,r,o)\in\mathcal{E}(o)} \mathbf{h}_q^{(t-1)}(s,x) \otimes \mathbf{w}_q(x,r,o) \right) \oplus \mathbf{h}_q^{(0)}(s,o), \tag{5.3}$$

where $\mathcal{E}(o)$ represents all edges with o as the object entity, i.e., $(*,*,o)$. [154] prove that T iterations of the generalized Bellman-Ford is equal to Eq. (5.1) with a max path length of T.

5.2.4 NBFNet

[154] extend Eq. (5.2) via the inclusion of learnable parameters. $\mathbf{w}_q(x,r,o)$ is replaced with a learnable embedding $\mathbf{w}_q(r)$ for each relation r. A linear transformation is further included in the aggregation. It is formulated as the following where for convenience we set $\mathbf{h}_q^{(t)}(s,o) = \mathbf{h}_o^{(t)}$ and $\mathbf{w}_q(x,r,o) = \mathbf{w}_q(r)$:

$$\begin{aligned} \mathbf{h}_o^{(0)} &= \text{INDICATOR}(u,v,q), \\ \mathbf{h}_o^{(t)} &= \text{AGG}\Big(\Big\{ \text{MSG}(\mathbf{h}_x^{(t-1)}, \mathbf{w}_q(r)) \mid (x,r,o) \in \mathcal{E}(o) \Big\} \cup \{\mathbf{h}_o^{(0)}\} \Big). \end{aligned} \tag{5.4}$$

The representation of the source node $\mathbf{h}_s^{(0)}$ is initialized to a learnt embedding, $\mathbf{q}_r$, corresponding to the query relation r. For all other nodes ($o \neq s$), they learn a separate initial embedding. However in practice they simply initialize the other nodes to the $\mathbf{0}$ vector. For the AGG function they consider the sum, max, min and PNA operations. For the MSG function they consider the TransE [14], DistMult [127], and RotatE [104] operators. The final representation is passed to a score function f which is modeled via an MLP.

5.3 The Proposed Framework

In this section, we propose a new approach to improve the efficiency of path-based GNN models. Inspired by two observations in Section 5.3.1, we proposed a simple but effective distance-based pruning strategy. We then introduce a truncated version of the generalized Bellman-Ford algorithm that achieves the goal of our proposed pruning strategy. Finally, we describe a neural network model based on the truncated Bellman-Ford.

5.3.1 Motivation

In this subsection, we discuss the motivation behind our framework design. In particular, we suggest that the inefficiency of path-based GNNs is mainly due to two observations: (1) the aggregation of many empty messages and (2) the proliferation of redundant messages when the number of layers is large. Next, we detail our observations and how they inspire us to design a more efficient method.

Observation #1: Empty Messages. Most path-based GNNs aggregate empty messages that do not contain any path information. This has the effect of increasing the model complexity without any obvious benefit. We provide an illustrative example. In Figure 5.1a, during the first iteration, node 7 will try to aggregate path information from node 6. However, all node representations, outside of the source, are initialized to zero ("empty messages"). Hence, a non-informative "empty message" will be passed to node 7 from node 6. In fact, in the first iteration, only the 1-hop neighbors of the source aggregate non-empty messages which contains information on paths with length 1. Only after two iterations will node 6 contain path information from the source. Therefore aggregating any messages before the third iteration will not lead to any path information for node 7. However, both NBFNet [154] and A*Net [152] will aggregate such messages, leading to increased complexity without any gain in additional path information. This observation suggests that *a node o of distance $dist(s, o)$ from the source can only aggregate path information from iteration $t = dist(s, o)$ onwards*.

Observation #2: Redundant Messages. Due to their design, *path-based GNNs with T layers can only learn representations for nodes within T hops of the source node*. However, since the time complexity of all existing methods is proportional to the number of layers, learning representations for nodes far from the source (i.e., distant nodes) can be very inefficient. In particular, as we discussed in Section 5.1, this mainly afflicts *target nodes closer to the source*. Again, we utilize Figure 5.1a for illustration. In the first two iterations the node 4 aggregates two paths including (source, 4) and (source, 3, 4). These paths provide significant information between the source and 4. Comparatively, in the 6-th iteration node 4 aggregates paths[1] of length 6, which reach further

[1]Strictly, these walks are not paths, as they contain repeated nodes and edges. In this paper, we follow the convention of the path-based GNN papers to loosely call them paths.

nodes and return to node 4. Since these paths already contain information present in shorter paths, little information is gained by aggregating them. Our empirical study in Section 5.4.3 also verifies that aggregating paths of longer length relative to the target node have little to no positive effect on performance.

These two observations suggest that the efficiency of path-based GNN methods is low when there are nodes of diverse distances to the source. We verify this by analyzing the distance distribution for all test samples on the WN18RR [30] dataset. For each sample we calculate the shortest path distance between both nodes and plot the distribution of the distances over all samples. The results are shown in Figure 5.2. We note that around 25% of samples have a shortest distance ≥ 5. To aggregate information for these distant nodes, it is necessary to set T to ≥ 5. In this case, nodes of larger distance will propagate empty messages for the first few iterations (Observation 1). Furthermore, about 35% of the samples have a shortest distance of 1. Such samples will aggregate redundant messages after a few iterations (Observation 2). **Our Design Goal**: The key to improving the efficiency of path-based GNNs is to modify their aggregation scheme. In particular, based on the aggregation scheme of path-based GNNs, all target nodes are aggregating paths with lengths ranging from 1 to T. Such paths contain many empty and redundant messages. To reduce the aggregation of those non-informative messages, we propose to customize the aggregations for each target node. Specifically, for close nodes, we do not aggregate long paths as they are redundant. For distant nodes, we do not aggregate short paths as they are empty. As such, we customize the aggregation process for each target node according to its distance from the source. Based on this intuition, we reformulate the path formulation, Eq. (5.1), as follows.

$$\mathbf{x}_q(s,o) = \bigoplus_{t=\text{dist}(s,o)}^{\text{dist}(s,o)+\delta} \bigoplus_{p\in P^t_{s,o}} \bigotimes_{i=1}^{|p|} w(e_i), \tag{5.5}$$

where $\delta \geq 0$ is an offset. The parameter δ can be considered as a form of path pruning as it controls the paths we aggregate relative to the shortest path distance. For example, when $\delta = 0$, it only aggregates those paths of the shortest distance for all node pairs. Empirical observations in Section 5.4.3 validate our use of pruning based on an offset δ.

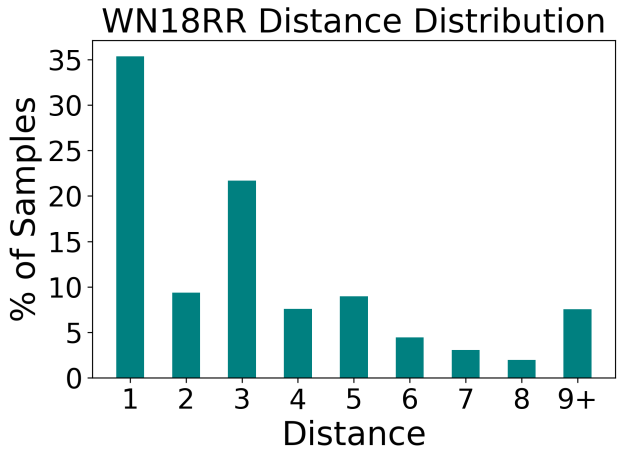


Figure 5.2 Test Distance Distribution for WN18RR.

Due to the high complexity of Eq. (5.5), it is not practical to directly calculate it. Hence, based on the generalized Bellman-Ford algorithm [9], we propose a truncated version of the Bellman-Ford algorithm for calculating Eq. (5.5) in a more efficient fashion.

5.3.2 Truncated Bellman-Ford

From our design goal, we are interested in capturing all paths of length $\text{dist}(s, o) \leq l \leq \text{dist}(s, o) + \delta$. To achieve this goal, for node o, we begin aggregating at iteration $t = \text{dist}(s, o)$ and stop aggregation after iteration $t = \text{dist}(s, o) + \delta$. This helps avoid aggregating empty messages before $\text{dist}(s, o)$-th iteration and redundant messages after $\text{dist}(s, o) + \delta$ iterations. However, during the iterations between $\text{dist}(s, o)$ and $\text{dist}(s, o) + \delta$, there are still potential empty messages. For example, any node v with the shortest distance to source larger than $\text{dist}(s, o) + \delta$ always contains empty messages during these iterations. Hence, to further avoid aggregating many empty messages, we only allow aggregation from a subset of the neighboring nodes of o. More formally, we formulate the above intuition into the following constrained edge set $C(s, o, t)$ through which node o aggregates information at iteration t.

$$C(s, o, t) = \begin{cases} \emptyset, \textbf{ if } t < \text{dist}(s, o) \text{ or } t > \text{dist}(s, o) + \delta \\ \{(v, r, o) \in \mathcal{E}(o) \mid \text{dist}(s, v) < \text{dist}(s, o) + \delta\}, \textbf{else} \end{cases} \tag{5.6}$$

Based on this constraint set of edges for node o, we update the generalized Bellman-Ford algorithm (Eq. 5.2) as follows where $C = C(s, o, t)$:

$$\mathbf{x}_q^{(t)}(s,o) = \left(\bigoplus_{(v,r,o)\in C} \mathbf{x}_q^{(t-1)}(s,v) \otimes \mathbf{w}_q(v,r,o) \right) \oplus \mathbf{x}_q^{(0)}(s,o). \tag{5.7}$$

The following theorem shows that the aggregation scheme proposed in Eq. (5.7) results in aggregation of the correct paths as described in Eq. (5.5).

Theorem 2. *Given a source node s, query q, and target node o, the final representation, $\mathbf{x}_q^F(s,o)$ only aggregates all path representations whose path length is between $dist(s,o)$ and $dist(s,o)+\delta$ for all $o \in V$. It therefore contains all information present in Eq.* (5.5) *such that,*

$$\mathbf{x}_q^F(s,o) = \bigoplus_{t=dist(s,o)}^{dist(s,o)+\delta} \bigoplus_{p\in P_{s,o}^t} \bigotimes_{i=1}^{|p|} w(e_i). \tag{5.8}$$

The detailed proof of Theorem 2 is provided in Appendix D.1. This design has the following advantages. **(1)** We don't begin aggregating messages until layer $t = \text{dist}(s,o)$. This helps avoid the aggregation of many empty messages for nodes far from the source. **(2)** We stop aggregating messages at layer $t = \text{dist}(s,o) + \delta$. This ensures that for close nodes we don't aggregate many redundant messages. Furthermore, it ensures that we will always aggregate paths of $\delta + 1$ different lengths for all target nodes regardless of their distance from the source. **(3)** In Section D.2.2, we demonstrate that the complexity of this design is *independent of the number of layers*, allowing for deep propagation.

An Illustrative Example. We given an example of the effect of constraints on propagation in Figure 5.1 where s = source. Figure 5.1a shows the involved nodes and edges over three iterations when $\delta = 1$. We observe that only a portion of the nodes and edges are involved at any one iteration. For example, at iteration 1 only the 1-hop neighbors and the edges connecting them to the source are involved. This is because they are the only nodes and edges able to receive any path information at that stage. Figure 5.1b details the update status of nodes by distance from the source node. We note how as the iteration increases the number of nodes updated shift to the right in groups of two. Furthermore since we only iterate for three iterations, the 4+ hop neighbors never update as there is no available path information for them until iteration 4.

5.3.3 Degree Messages

An effect of pruning paths, especially with low δ, is that it can lead to very few messages being aggregated. This is especially true for smaller or sparser graphs. One consequence of few messages being aggregated is that it can make it difficult for a node to discern the properties of its neighborhood (e.g. degree). We give an example of node 4 in Figure 5.1. For each of the first 2 iterations, it only aggregates messages from 2/4 of it's neighbors. As such, it never aggregates messages from all its neighbors at the same iteration. This can lead to a failure of node 4 to properly discern it's degree, as the number of non-empty messages in each iteration is only a portion of the overall degree. Since the degree is known to be an important factor in link prediction [81, 2], we want to preserve the degree information for all nodes.

In order to preserve the degree information for each node, we consider encoding the degree via the use of pseudo messages. Specifically, we want to add enough messages such that the total number of messages aggregated for a node o is equivalent to its degree. We refer to such messages as *degree messages*. Going back to our example in Figure 5.1, for node 4 at iteration 1 and 2 we would add 2 degree messages so that the total number of messages is 4. Formally, we denote the degree of a node o as b_o. The number of messages to add at iteration t is given by $\rho_o = b_o - |C(s, o, t)|$.

For the value of the messages, we learn a separate embedding denoted as $\mathbf{x}_{\text{deg}}^{(t)}$ that is the same across all nodes. Since the value of each message is the same we can avoid explicitly aggregating each degree message individually. Instead, we just aggregate one message that is equal to the number of degree messages multiplied by the degree embedding,

$$\mathbf{x}_{\text{deg}}^{(t)}(s, o) = \rho_o \cdot \mathbf{x}_{\text{deg}}^{(t)}, \tag{5.9}$$

where $\mathbf{x}_{\text{deg}}^{(t)}(s, o)$ is the value of the degree message for node o at iteration t. This edge is then added to the set of messages to be aggregated, $C(s, o, t)$. Since this is equivalent to computing and aggregating only one edge, it has no effect on the model complexity. Experimental results in Section 5.4.4 validate the effectiveness of degree messages.

5.3.4 GNN Formulation

We follow similar conventions to NBFNet when converting Eq. (5.6) and Eq. (5.7) to a GNN. We denote the embedding of a source node s and arbitrary target node o as $\mathbf{x}_q(s, o)$. We further represent the indicator query embeddings as $\mathbf{x}_q$ and the layer-wise relation embeddings as $\mathbf{x}_r^{(t)}$.

We utilize the INDICATOR function described in Section 5.2.4, PNA [26] for the AGGREGATE function, and DistMult [127] for the MSG function. The probability of a link existing between a source-target pair is determined via a score function f. Both the final representation of the pair and the query embedding are given as input. The output of f is then passed to a sigmoid to produce a probability,

$$p(s, o) = \sigma\left(f\left(\mathbf{x}_q^{\mathrm{F}}(s, o),\, \mathbf{x}_q\right)\right), \tag{5.10}$$

where $\mathbf{x}_q^{\mathrm{F}}(s, o)$ is the final pair representation. The full algorithm is detailed in Appendix D.2.1. We run a total of T layers. We further show in in Appendix D.2.2 that time complexity is independent of the number of layers. This enables TAGNet to propagate for more layers than existing path-based GNNs.

Furthermore, due to its general design, TAGNet can also be integrated with other efficiency-minded methods like A*Net. This is described in more detail in Appendix D.2.3. Extensive experiments in Sections 5.4.1 and 5.4.2 also demonstrate that combining both methods can significantly reduce the number of messages propagated by A*Net without sacrificing performance.

5.3.5 Target-Specific δ

A drawback of our current design is that we assume a single offset δ for all possible node pairs. However, for some pairs we may want to consider propagating more or less iterations. For example, in Figure 5.1 we may only want to consider $\delta = 0$ for the target node 2 due to the limited number of paths connecting it to the source. However for node 4, which is concentrated in a denser portion of the subgraph, we may want to consider a higher value of δ such as 1 or 2 to capture more path information. We next detail our method for achieving this.

5.3.5.1 Target-Specific δ via Attention

A target-specific δ can be attained by realizing the connection between the hidden representations and the value of δ. Let's denote the value of the hyperparameter δ as $\hat{\delta}$. For a source-target node pair (s, o), we only aggregate paths from length $\text{dist}(s, o)$ to $\text{dist}(s, o) + \hat{\delta}$. At iteration $t = \text{dist}(s, o)$ we aggregate paths of length $\text{dist}(s, o)$ and at iteration $t = \text{dist}(s, o) + 1$ only those paths of length $\text{dist}(s, o) + 1$, and so on until $t = \text{dist}(s, o) + \hat{\delta}$. The set of hidden representations for a node pair is as follows where for convenience we represent $\mathbf{x}_q(s, o)$ as $\mathbf{x}_{(s,o)}$:

$$\text{Hiddens}(s, o) = \left[\mathbf{x}_{(s,o)}^{\text{dist}(s,o)}, \cdots, \mathbf{x}_{(s,o)}^{(\text{dist}(s,o)+\hat{\delta})}\right]. \tag{5.11}$$

The first hidden representation only contains paths of shortest length and therefore corresponds to $\delta = 0$. Since the paths accumulate over hidden representations via a self-loop, $\mathbf{x}_{(s,o)}^{(\text{dist}(s,o)+1)}$ contains all paths of length $\text{dist}(s, o)$ and $\text{dist}(s, o) + 1$, corresponding to $\delta = 1$. As such, the final hidden representation is equivalent to $\delta = \hat{\delta}$. Therefore, choosing a target-specific δ is achieved by selecting one of the hidden representations as the final representation.

We utilize attention to determine which value of δ is best for a specific target node. This is formulated as the following:

$$\mathbf{x}_{(s,o)}^{\text{F}} = \sum_{\delta=0}^{\hat{\delta}} \alpha_{(s,o)}^{\delta} \mathbf{x}_{(s,o)}^{(\text{dist}(s,o)+\delta)}, \tag{5.12}$$

where $\alpha_{(s,o)}^{\delta}$ is the corresponding attention weight for the hidden representation $\mathbf{x}_{(s,o)}^{(\text{dist}(s,o)+\delta)}$. For each possible value of δ, $\alpha_{(s,o)}^{\delta}$ is given by:

$$\tilde{\alpha}_{(s,o)}^{\delta} = g\left(\mathbf{x}_{(s,o)}^{(\text{dist}(s,o)+\delta)}, \mathbf{x}_q\right)$$
$$\alpha_{(s,o)}^{\delta} = \text{Softmax}(\tilde{\alpha}_{(s,o)}^{\delta}).$$

We model g as an MLP that takes both the hidden representation and the query embedding as input. Taking inspiration from A*Net [152], we conjecture that a well-learned score function can help determine which representations are better than others. As such, we further consider modeling g as its own function or having it share parameters with the score function f, Eq. (5.10). Lastly, we show in Appendix D.2.2 that the time complexity is unchanged when using a target-specific δ.

Table 5.1 Transductive Results. Best results are in **bold** and the 2nd best underlined.

Method Type	Method	FB15k-237			WN18RR		
		MRR	Hits@1	Hits@10	MRR	Hits@1	Hits@10
Embeddings	TransE	0.294	-	0.465	0.226	-	0.501
	DistMult	0.241	0.155	0.419	0.43	0.39	0.49
	ComplEx	0.247	0.158	0.428	0.44	0.41	0.51
GNNs	R-GCN	0.273	0.182	0.456	0.402	0.345	0.494
	CompGCN	0.355	0.264	0.535	0.479	0.443	0.546
Path-Based	DRUM	0.343	0.255	0.516	0.486	0.425	0.586
	RED-GNN	0.374	0.283	0.558	0.533	0.485	0.624
	AdaProp	0.392	0.309	0.555	0.553	0.502	0.652
	NBFNet	0.415	0.321	0.599	0.551	0.497	**0.666**
	A*Net	0.414	0.324	0.592	0.547	0.490	0.658
TAGNet	+ A*Net	0.409	0.323	0.577	0.555	0.502	0.657
	Fixed δ	**0.421**	**0.328**	**0.602**	0.562	0.509	**0.667**
	Specific δ	0.417	**0.328**	0.592	**0.565**	**0.513**	**0.667**

Table 5.2 Inductive Results (evaluated with Hits@10). Ours results are averaged over 5 runs.

Method	FB15k-237				WN18RR			
	v1	v2	v3	v4	v1	v2	v3	v4
NeuralLP	0.468	0.586	0.571	0.593	0.772	0.749	0.476	0.706
DRUM	0.474	0.595	0.571	0.593	0.777	0.747	0.477	0.702
GraIL	0.429	0.424	0.424	0.389	0.760	0.776	0.409	0.687
RED-GNN	0.483	0.629	0.603	0.621	0.799	0.780	0.524	0.721
AdaProp	0.470	0.651	0.620	0.614	0.798	**0.836**	**0.582**	0.732
NBFNet	**0.607**	**0.704**	0.667	**0.668**	**0.826**	0.798	0.568	0.694
A*Net	0.535	0.638	0.610	0.630	0.810	0.803	0.544	0.743
TAGNet + A*Net	0.541	0.646	0.604	0.623	0.813	0.805	0.535	**0.745**
TAGNet (fixed δ)	0.596	0.700	**0.677**	0.666	0.816	0.796	0.534	0.734
TAGNet (specific δ)	0.596	0.698	0.675	0.661	0.818	0.803	0.544	0.737

5.4 Experiment

In this section, we evaluate the effectiveness of our proposed framework on KGC under both the transductive and inductive settings. We also empirically analyze the efficiency and conduct ablation studies on each component. The experimental details are listed in Appendix D.3. We note that for a fair comparison between path-based GNNs, we run each model using 6 layers and a hidden dimension of 32 as is done in both [154] and [152]. Please see Appendix D.3.2 for more details.

5.4.1 Effectiveness of TAGNet

In this subsection, we present the results of TAGNet compared with baselines on both transductive and inductive settings. We further detail the results when combining TAGNet with A*Net.

Transductive Setting: The results on the transductive setting are shown in Table 5.1. We observe that TAGNet achieves strong performance with just a fixed δ. In particular, it outperforms A*Net and AdaProp on most metrics. Also compared to NBFnet, which doesn't utilize pruning, TAGNet achieves comparable or even stronger performance. This indicates that the proposed pruning strategy mostly reduces redundant aggregations that do not impair the models effectiveness.

Inductive Setting: Table 5.2 shows the results on the inductive setting. TAGNet achieves strong performance on both datasets. In particular, it achieves comparable performance to the non-pruning version of NBFNet. Furthermore, TAGNet significantly outperforms A*Net and AdaProp on the FB15k-237 splits, demonstrating the advantage of the proposed pruning strategy.

TAGNet + A*Net: We further test combining the pruning strategy of both TAGNet and A*Net together (see Appendix D.2.3 for more details). Compared to A*Net, we observe that TAGNet+A*Net achieves comparable if not better performance under all settings despite aggregating much fewer messages (see subsection 5.4.2). This suggests that the pruning strategy in A*Net fails to prune many irrelevant paths, allowing TAGNet to work complementary to it.

5.4.2 Efficiency of TAGNet

In this subsection, we empirically evaluate the efficiency of our model against NBFNet. Specifically, we compare the mean number of messages aggregated per sample during training.

Figure 5.3 shows the % decrease in the number of messages of TAGNet as compared to NBFNet. All models are fit with 6 layers. We observe two trends. The first is that both FB15k-237 datasets follow a similar relationship that is close to what's expected of the worst-case complexity detailed in Appendix D.2.2. On the other hand, the WN18RR datasets pass much fewer messages as they hover above 90% for all δ. This is likely because WN18RR is a very sparse graph. This gives TAGNet plenty of opportunities to prune paths.

We further compare the efficiency of just A*Net and A*Net + TAGNet. As before, we calculate

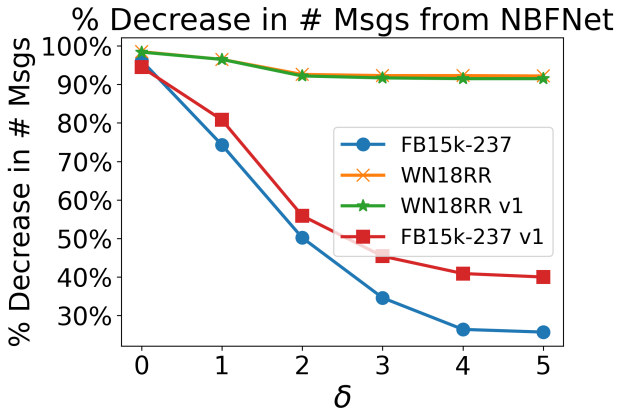


Figure 5.3 % Decrease in NBFNet Messages.

the total number of messages passed for both methods. We fix $\delta = 2$. Table 5.3 show the % decrease in the number of messages when utilizing both techniques compared to just A*Net. We observe a large reduction in both the inductive and transductive setting. Since the performance of A*Net + TAGNet is on par with just A*Net, it suggests that A*Net fails to prune many unneeded messages that do not improve performance. Furthermore, we find that the reduction in the number of messages becomes more pronounced with more layers, suggesting that TAGNet is even more useful when deep propagation is necessary.

Table 5.3 % Decrease in # Msgs for A*Net vs. A*Net + TAGNet.

Dataset	**6 Layers**	**7 Layers**	**8 Layers**
FB15k-237	39%	51%	59%
FB15k-237 v1	30%	44%	66%
WN18RR	10%	17%	26%
WN18RR v1	25%	37%	46%

5.4.3 Effect of δ

In this subsection, we evaluate the effect of the offset δ on TAGNet test performance (w/o the target-specific setting). We fix the number of layers at 6 and vary δ from 0 to 5. We report results for both the transductive and inductive settings in Figures 5.4 and 5.5, respectively. For the inductive setting, we chose version v1 of both datasets as the representative datasets. For both transductive datasets, we find that the performance plateaus at $\delta = 2$. A similar trend is observed for FB15k-237 v1. Interestingly, for WN18RR v1,the performance is constant when varying δ. This suggests that

for some datasets almost all of the important information is concentrated in paths of the shortest length.

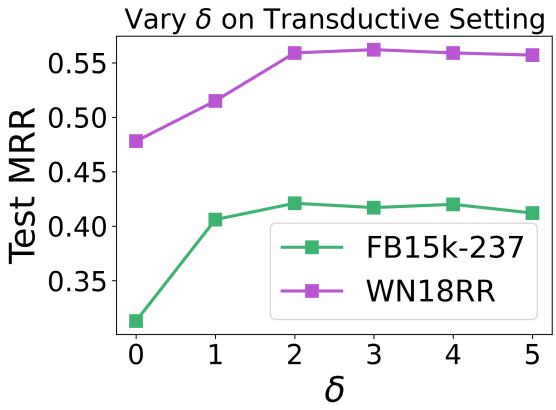


Figure 5.4 Performance varying δ on Transductive setting.

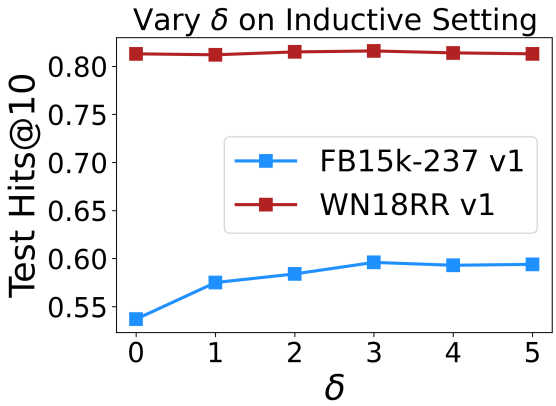


Figure 5.5 Performance varying δ on Inductive setting.

5.4.4 Effect of Degree Messages

We demonstrate the effect of the degree messages described in Section 5.3.3. Table 5.4 shows the performance of TAGNet when trained with and without degree messages. We report the performance on all of the inductive splits for both FB15k-237 and WN18RR. Interestingly, we observe that while there is a consistent gain on FB15k-237, it often hurts performance on WN18RR. This may imply that preserving the degree information of each node is more important on FB15k-237 than WN18RR.

Table 5.4 Effect of Degree Messages on Inductive Splits.

Dataset	Split	w/o Msgs	with Msgs
FB15k-237	V1	0.594	**0.596**
	V2	0.684	**0.698**
	V3	0.653	**0.675**
	V4	0.648	**0.661**
WN18RR	V1	0.815	**0.818**
	V2	**0.803**	0.781
	V3	**0.544**	0.465
	V4	**0.737**	0.718

5.5 Related Work

We give a brief overview of different types of KGC methods. **(1) Embedding-Based Methods**: Such methods are concerned with modeling the interactions of entity and relation embeddings. TransE [14] models each fact as translation in the embedding space while DistMult [127] scores each fact via a bilinear diagonal function. ComplEx [112] extends DistMult by further modeling the embeddings in the complex space. Lastly, Nodepiece [36] attempts to improve the efficiency of embedding-based KGC methods by representing each entity embedding as a combination of a smaller set of subword embeddings. Since this method concerns embedding-based techniques, it is orthogonal to our work. **(2) GNN-Based Methods**: GNN methods extend traditional GNNs by further considering the relational information. CompGCN [113] encodes each message as a combination of neighboring entity-relation pairs via the use of compositional function. RGCN [100] instead considers a relation-specific transformation matrix to integrate the relation information. **(3) Path-Based Methods**: Path-based methods attempt to leverage the path information connecting two entities to perform KGC. NeuralLP [128] and DRUM [96] learn to weight different paths by utilizing logical rules. More recently, NBFNet [154] considers path information by learning a parameterized version of the Bellman-Ford algorithm. A similar framework, RED-GNN [143] also attempts to take advantage of dynamic programming to aggregate path information. Both A*Net [152] and AdaProp [144] attempt to prove upon the efficiency of the previous methods by learning which nodes to propagate to.

5.6 Conclusion

In this paper we identify two intrinsic limitations of path-based GNNs that affect the efficiency and representation quality. We tackle these issues by introducing a new method, TAGNet, which is able to efficiently propagate path information. This is realized by only aggregating paths in a fixed window for each source-target pair. We demonstrate that the complexity of TAGNet is independent of the number of layers. For future work, we plan on exploring methods to capture path information without having to perform a separate round of propagation for every individual source node.

CHAPTER 6

CONCLUSION AND FUTURE DIRECTIONS

In this work we propose enhancing link prediction from multiple perspectives through a data-centric framework. To do so, we first study the evaluation setting of link prediction, showing that it is unrealistic. We then promote a new evaluation procedure which corresponds to a more realistic evaluation of link prediction. Next, we propose a new method, LPFormer, that can efficiently model the factors necessary for LP in a data-driven way. The design of LPFormer is based on our empirical understanding of LP performance. We then focus on KG completion, seeking to mitigate problems with bias and efficiency. We first study the problem of degree bias in KGC, finding that it differs from traditional degree bias in node classification. We then propose a new data augmentation method to alleviate this bias. Lastly, we study the inefficiency of path-based GNNs used for KGC. We find that much computation is spend on redundant or empty messages. We then propose a new method for reducing these messages.

In the future, we plan to further explore LP and Graph ML in various ways:

1. **Generalizable LP**: To properly work with real-world data, it's necessary to design methods that can generalize to out-of-distribution (OOD) samples. Recent work has shown [92] has shown that for LP, methods often struggle to properly generalize across covariate distribution shifts. However, there is little work exploring how to practically overcome these issues and design methods that can generalize. Furthermore, it's unclear how current methods can generalize to links that follow different structural patterns. I plan to explore both directions in future work.
2. **Trustworthy LP**: It has been shown that AI is susceptible to malicious attacks on the input given to models. It is necessary to safeguard against such potential attackers given its potential to cause great harm in safety-critical and real-world applications. Recent work [60] has further shown that for the task of LP, diffusion models can be helpful in protecting against certain types of adversarial attacks. However, despite the real-world use of such protection, it is still relatively unexplored. We therefore plan on continuing this line of research by exploring how current LP models can protect against a more diverse set of adversarial attacks and whether we can design

more efficient models to counterattack such attacks.

3. **Foundation models for graph data.** Recently, we've seen the emergence of "foundation models" (FMs), which are a single model that can be used on a broad spectrum of downstream tasks and datasets. The key advantage of FMs are that they allow us to sidestep the need to train a new model for each dataset and task. However, despite the success in other fields like CV and NLP, FMs have yet to take hold in the graph domain. Current graph foundation models (GFMs) are limited, only working for specific domains (e.g., KG reasoning) or with natural language node features. Recent work of ours has explored creating LP models that can generalize across different datasets and domains [60], a key requirement for any FM. I plan to continue developing GFMs that can be applied to graphs of various domains and for multiple different tasks. In order to do this, they are several key questions that we need to answer: How to handle the heterogeneity of the initial node features, not only in size but in semantic meaning? How do we contend with the fact that different tasks (e.g., node classification and LP) and domains require different inductive biases and therefore different final representations to perform optimally? Can we learn to extract a shared set of information that is useful across different tasks and domains? I argue that is necessary to answer these questions if we hope to create meaningful GFMs.

BIBLIOGRAPHY

[1] Khushnood Abbas, Alireza Abbasi, Shi Dong, Ling Niu, Laihang Yu, Bolun Chen, Shi-Min Cai, and Qambar Hasan. Application of network link prediction in drug discovery. *BMC bioinformatics*, 22:1–21, 2021.

[2] Lada A Adamic and Eytan Adar. Friends and neighbors on the web. *Social networks*, 25(3):211–230, 2003.

[3] Mehdi Ali, Max Berrendorf, Charles Tapley Hoyt, Laurent Vermue, Mikhail Galkin, Sahand Sharifzadeh, Asja Fischer, Volker Tresp, and Jens Lehmann. Bringing light into the dark: A large-scale evaluation of knowledge graph embedding models under a unified framework. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(12):8825–8845, 2021.

[4] Saadullah Amin, Stalin Varanasi, Katherine Ann Dunfield, and Günter Neumann. Lowfer: Low-rank bilinear pooling for link prediction. In *International Conference on Machine Learning*, pages 257–268. PMLR, 2020.

[5] Reid Andersen, Fan Chung, and Kevin Lang. Local graph partitioning using pagerank vectors. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*, pages 475–486. IEEE, 2006.

[6] Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *3rd International Conference on Learning Representations, ICLR 2015*, 2015.

[7] Ivana Balažević, Carl Allen, and Timothy M Hospedales. Tucker: Tensor factorization for knowledge graph completion. *arXiv preprint arXiv:1901.09590*, 2019.

[8] Albert-Laszlo Barabâsi, Hawoong Jeong, Zoltan Néda, Erzsebet Ravasz, Andras Schubert, and Tamas Vicsek. Evolution of the social network of scientific collaborations. *Physica A: Statistical mechanics and its applications*, 311(3-4):590–614, 2002.

[9] John S Baras and George Theodorakopoulos. Path problems in networks. *Synthesis Lectures on Communication Networks*, 3(1):1–77, 2010.

[10] Aleksandar Bojchevski, Johannes Gasteiger, Bryan Perozzi, Amol Kapoor, Martin Blais, Benedek Rózemberczki, Michal Lukasik, and Stephan Günnemann. Scaling graph neural networks with approximate pagerank. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2464–2473, 2020.

[11] Stephen Bonner, Ian P Barrett, Cheng Ye, Rowan Swiers, Ola Engkvist, Charles Tapley Hoyt, and William L Hamilton. Understanding the performance of knowledge graph embeddings in drug discovery. *Artificial Intelligence in the Life Sciences*, page 100036, 2022.

[12] JC de Borda. M'emoire sur les' elections au scrutin. *Histoire de l'Acad'emie Royale des Sciences*, 1781.

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

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

[15] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

[16] Andrei Z Broder. On the resemblance and containment of documents. In *Proceedings. Compression and Complexity of SEQUENCES 1997 (Cat. No. 97TB100171)*, pages 21–29. IEEE, 1997.

[17] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? In *International Conference on Learning Representations*, 2022.

[18] Yixin Cao, Xiang Wang, Xiangnan He, Zikun Hu, and Tat-Seng Chua. Unifying knowledge graph learning and recommendation: Towards a better understanding of user preferences. In *The world wide web conference*, pages 151–161, 2019.

[19] Luigi Carratino, Moustapha Cissé, Rodolphe Jenatton, and Jean-Philippe Vert. On mixup regularization. *arXiv preprint arXiv:2006.06049*, 2020.

[20] Benjamin Paul Chamberlain, Sergey Shirobokov, Emanuele Rossi, Fabrizio Frasca, Thomas Markovich, Nils Hammerla, Michael M Bronstein, and Max Hansmire. Graph neural networks for link prediction with subgraph sketching. *arXiv preprint arXiv:2209.15486*, 2022.

[21] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[22] Jinsong Chen, Kaiyuan Gao, Gaichao Li, and Kun He. Nagphormer: A tokenized graph transformer for node classification in large graphs. In *The Eleventh International Conference on Learning Representations*, 2022.

[23] Sanxing Chen, Xiaodong Liu, Jianfeng Gao, Jian Jiao, Ruofei Zhang, and Yangfeng Ji. Hitter: Hierarchical transformers for knowledge graph embeddings. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10395–10407, 2021.

[24] Fan Chung. The heat kernel as the pagerank of a graph. *Proceedings of the National Academy of Sciences*, 104(50):19735–19740, 2007.

[25] Gonçalo M Correia, Vlad Niculae, and André FT Martins. Adaptively sparse transformers. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2174–2184, 2019.

[26] Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. *Advances in Neural Information Processing Systems*, 33:13260–13271, 2020.

[27] Nur Nasuha Daud, Siti Hafizah Ab Hamid, Muntadher Saadoon, Firdaus Sahran, and Nor Badrul Anuar. Applications of link prediction in social networks: A review. *Journal of Network and Computer Applications*, 166:102716, 2020.

[28] Jesse Davis and Mark Goadrich. The relationship between precision-recall and roc curves. In *Proceedings of the 23rd international conference on Machine learning*, pages 233–240, 2006.

[29] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

[30] Tim Dettmers, Pasquale Minervini, Pontus Stenetorp, and Sebastian Riedel. Convolutional 2d knowledge graph embeddings. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[31] Yuxiao Dong, Reid A Johnson, Jian Xu, and Nitesh V Chawla. Structural diversity and homophily: A study across more than one hundred big networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 807–816, 2017.

[32] Vijay Prakash Dwivedi, Chaitanya K Joshi, Anh Tuan Luu, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *Journal of Machine Learning Research*, 24(43):1–48, 2023.

[33] Chantat Eksombatchai, Pranav Jindal, Jerry Zitao Liu, Yuchen Liu, Rahul Sharma, Charles Sugnet, Mark Ulrich, and Jure Leskovec. Pixie: A system for recommending 3+ billion items to 200+ million users in real-time. In *Proceedings of the 2018 world wide web conference*, pages 1775–1784, 2018.

[34] Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. In *Proceedings of the 8th International Conference on Learning Representations (ICLR)*, 2020.

[35] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. *Discrete mathematics & theoretical computer science*, (Proceedings), 2007.

[36] Mikhail Galkin, Etienne Denis, Jiapeng Wu, and William L Hamilton. Nodepiece: Compositional and parameter-efficient representations of large knowledge graphs. In *International Conference on Learning Representations*, 2021.

[37] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

[38] David F Gleich, Kyle Kloster, and Huda Nassar. Localization in seeded pagerank. *arXiv preprint arXiv:1509.00016*, 2015.

[39] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864, 2016.

[40] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. On calibration of modern neural networks. In *International conference on machine learning*, pages 1321–1330. PMLR, 2017.

[41] Pankaj Gupta, Ashish Goel, Jimmy Lin, Aneesh Sharma, Dong Wang, and Reza Zadeh. Wtf: The who to follow service at twitter. In *Proceedings of the 22nd international conference on World Wide Web*, pages 505–514, 2013.

[42] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[43] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020.

[44] Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *Advances in neural information processing systems*, 33:22118–22133, 2020.

[45] Hong Huang, Jie Tang, Lu Liu, JarDer Luo, and Xiaoming Fu. Triadic closure pattern analysis and prediction in social networks. *IEEE Transactions on Knowledge and Data Engineering*, 27(12):3374–3389, 2015.

[46] Zan Huang, Xin Li, and Hsinchun Chen. Link prediction approach to collaborative filtering. In *Proceedings of the 5th ACM/IEEE-CS joint conference on Digital libraries*, pages 141–142, 2005.

[47] Zexi Huang, Mert Kosan, Arlei Silva, and Ambuj Singh. Link prediction without graph neural networks. *arXiv preprint arXiv:2305.13656*, 2023.

[48] Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

[49] Leo Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, 1953.

[50] Leo Katz. A new status index derived from sociometric analysis. *Psychometrika*, 18(1):39–43, 1953.

[51] Jinwoo Kim, Dat Nguyen, Seonwoo Min, Sungjun Cho, Moontae Lee, Honglak Lee, and Seunghoon Hong. Pure transformers are powerful graph learners. *Advances in Neural Information Processing Systems*, 35:14582–14595, 2022.

[52] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[53] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[54] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[55] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

[56] Sadamori Kojaku, Jisung Yoon, Isabel Constantino, and Yong-Yeol Ahn. Residual2vec: Debiasing graph embedding with random graphs. *Advances in Neural Information Processing Systems*, 34, 2021.

[57] István A Kovács, Katja Luck, Kerstin Spirohn, Yang Wang, Carl Pollis, Sadie Schlabach, Wenting Bian, Dae-Kyum Kim, Nishka Kishore, Tong Hao, et al. Network-based prediction of protein interactions. *Nature communications*, 10(1):1240, 2019.

[58] Devin Kreuzer, Dominique Beaini, Will Hamilton, Vincent Létourneau, and Prudencio Tossou. Rethinking graph transformers with spectral attention. *Advances in Neural Information Processing Systems*, 34:21618–21629, 2021.

[59] Guangyao Li, Zequn Sun, Lei Qian, Qiang Guo, and Wei Hu. Rule-based data augmentation for knowledge graph embedding. *AI Open*, 2:186–196, 2021.

[60] Hang Li, Wei Jin, Geri Skenderi, Harry Shomer, Wenzhuo Tang, Wenqi Fan, and Jiliang

Tang. Sub-graph based diffusion model for link prediction. In *The Third Learning on Graphs Conference*.

[61] Juanhui Li, Harry Shomer, Haitao Mao, Shenglai Zeng, Yao Ma, Neil Shah, Jiliang Tang, and Dawei Yin. Evaluating graph neural networks for link prediction: Current pitfalls and new benchmarking. *arXiv preprint arXiv:2306.10453*, 2023.

[62] Juanhui Li, Harry Shomer, Haitao Mao, Shenglai Zeng, Yao Ma, Neil Shah, Jiliang Tang, and Dawei Yin. Evaluating graph neural networks for link prediction: Current pitfalls and new benchmarking. *Advances in Neural Information Processing Systems*, 36, 2024.

[63] Pan Li, Yanbang Wang, Hongwei Wang, and Jure Leskovec. Distance encoding: Design provably more powerful neural networks for graph representation learning. *Advances in Neural Information Processing Systems*, 33:4465–4478, 2020.

[64] Ren Li, Yanan Cao, Qiannan Zhu, Guanqun Bi, Fang Fang, Yi Liu, and Qian Li. How does knowledge graph embedding extrapolate to unseen data: a semantic evidence view. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2022*, 2022.

[65] David Liben-Nowell and Jon Kleinberg. The link prediction problem for social networks. In *Proceedings of the twelfth international conference on Information and knowledge management*, pages 556–559, 2003.

[66] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[67] Weijie Liu, Peng Zhou, Zhe Zhao, Zhiruo Wang, Qi Ju, Haotang Deng, and Ping Wang. K-bert: Enabling language representation with knowledge graph. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 2901–2908, 2020.

[68] Zemin Liu, Trung-Kien Nguyen, and Yuan Fang. Tail-gnn: Tail-node graph neural networks. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1109–1119, 2021.

[69] Linyuan Lü and Tao Zhou. Link prediction in complex networks: A survey. *Physica A: statistical mechanics and its applications*, 390(6):1150–1170, 2011.

[70] Sijie Mai, Shuangjia Zheng, Yuedong Yang, and Haifeng Hu. Communicative message passing for inductive relation reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 4294–4302, 2021.

[71] Haitao Mao, Zhikai Chen, Wei Jin, Haoyu Han, Yao Ma, Tong Zhao, Neil Shah, and Jiliang Tang. Demystifying structural disparity in graph neural networks: Can one size fit all? *Advances in Neural Information Processing Systems*, 36, 2024.

[72] Haitao Mao, Juanhui Li, Harry Shomer, Bingheng Li, Wenqi Fan, Yao Ma, Tong Zhao, Neil Shah, and Jiliang Tang. Revisiting link prediction: A data perspective, 2023.

[73] Sameen Maruf, André FT Martins, and Gholamreza Haffari. Selective attention for context-aware neural machine translation. In *Proceedings of NAACL-HLT*, pages 3092–3102, 2019.

[74] Aditya Krishna Menon and Charles Elkan. Link prediction via matrix factorization. In *Joint european conference on machine learning and knowledge discovery in databases*, pages 437–452. Springer, 2011.

[75] Grégoire Mialon, Dexiong Chen, Margot Selosse, and Julien Mairal. Graphit: Encoding graph structure in transformers. *arXiv preprint arXiv:2106.05667*, 2021.

[76] Aisha Mohamed, Shameem Parambath, Zoi Kaoudi, and Ashraf Aboulnaga. Popularity agnostic evaluation of knowledge graph embeddings. In *Conference on Uncertainty in Artificial Intelligence*, pages 1059–1068. PMLR, 2020.

[77] Sameh K Mohamed, Vít Nováček, and Aayah Nounu. Discovering protein drug targets using knowledge graph embeddings. *Bioinformatics*, 36(2):603–610, 2020.

[78] Luis Müller, Mikhail Galkin, Christopher Morris, and Ladislav Rampášek. Attending to graph transformers. *arXiv preprint arXiv:2302.04181*, 2023.

[79] Yohsuke Murase, Hang-Hyun Jo, János Török, János Kertész, and Kimmo Kaski. Structural transition in social networks: The role of homophily. *Scientific reports*, 9(1):4310, 2019.

[80] Huda Nassar, Kyle Kloster, and David F Gleich. Strong localization in personalized pagerank vectors. In *Proceedings of the 12th International Workshop on Algorithms and Models for the Web Graph-Volume 9479*, pages 190–202, 2015.

[81] Mark EJ Newman. Clustering and preferential attachment in growing networks. *Physical review E*, 64(2):025102, 2001.

[82] Tu Dinh Nguyen, Dat Quoc Nguyen, Dinh Phung, et al. A novel embedding model for knowledge base completion based on convolutional neural network. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 327–333, 2018.

[83] Maximilian Nickel, Xueyan Jiang, and Volker Tresp. Reducing the rank in relational factorization models by including observable patterns. *Advances in Neural Information Processing Systems*, 27, 2014.

[84] Maximilian Nickel, Volker Tresp, and Hans-Peter Kriegel. A three-way model for collective learning on multi-relational data. In *Icml*, 2011.

[85] Vardaan Pahuja, Boshi Wang, Hugo Latapie, Jayanth Srinivasa, and Yu Su. A retrieve-and-read framework for knowledge graph link prediction. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pages 1992–2002, 2023.

[86] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[87] Geoff Pleiss, Manish Raghavan, Felix Wu, Jon Kleinberg, and Kilian Q Weinberger. On fairness and calibration. *Advances in neural information processing systems*, 30, 2017.

[88] Ladislav Rampášek, Michael Galkin, Vijay Prakash Dwivedi, Anh Tuan Luu, Guy Wolf, and Dominique Beaini. Recipe for a general, powerful, scalable graph transformer. *Advances in Neural Information Processing Systems*, 35:14501–14515, 2022.

[89] Sylvestre-Alvise Rebuffi, Sven Gowal, Dan Andrei Calian, Florian Stimberg, Olivia Wiles, and Timothy A Mann. Data augmentation can improve robustness. *Advances in Neural Information Processing Systems*, 34:29935–29948, 2021.

[90] Jiawei Ren, Cunjun Yu, Xiao Ma, Haiyu Zhao, Shuai Yi, et al. Balanced meta-softmax for long-tailed visual recognition. *Advances in neural information processing systems*, 33:4175–4186, 2020.

[91] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. Bpr: Bayesian personalized ranking from implicit feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, pages 452–461, 2009.

[92] Jay Revolinsky, Harry Shomer, and Jiliang Tang. Understanding the generalizability of link predictors under distribution shifts on graphs. *arXiv preprint arXiv:2406.08788*, 2024.

[93] Andrea Rossi, Denilson Barbosa, Donatella Firmani, Antonio Matinata, and Paolo Merialdo. Knowledge graph embedding for link prediction: A comparative analysis. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 15(2):1–49, 2021.

[94] Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-scale attributed node embedding. *Journal of Complex Networks*, 9(2):cnab014, 2021.

[95] Daniel Ruffinelli, Samuel Broscheit, and Rainer Gemulla. You can teach an old dog new tricks! on training knowledge graph embeddings. In *International Conference on Learning Representations*.

[96] Ali Sadeghian, Mohammadreza Armandpour, Patrick Ding, and Daisy Zhe Wang. Drum: End-to-end differentiable rule mining on knowledge graphs. *Advances in Neural Information Processing Systems*, 32, 2019.

[97] Tara Safavi and Danai Koutra. Codex: A comprehensive knowledge graph completion benchmark. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 8328–8350, 2020.

[98] Tara Safavi and Danai Koutra. Codex: A comprehensive knowledge graph completion benchmark. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 8328–8350, 2020.

[99] Takaya Saito and Marc Rehmsmeier. The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. *PloS one*, 10(3):e0118432, 2015.

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

[101] Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

[102] Balasubramaniam Srinivasan and Bruno Ribeiro. On the equivalence between positional node embeddings and structural graph representations. In *International Conference on Learning Representations*, 2019.

[103] Arjun Subramonian, Jian Kang, and Yizhou Sun. Theoretical and empirical insights into the origins of degree bias in graph neural networks. *arXiv preprint arXiv:2404.03139*, 2024.

[104] Zhiqing Sun, Zhi-Hong Deng, Jian-Yun Nie, and Jian Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. In *International Conference on Learning Representations*, 2019.

[105] Zhiqing Sun, Shikhar Vashishth, Soumya Sanyal, Partha Talukdar, and Yiming Yang. A re-evaluation of knowledge graph completion methods. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 5516–5522, 2020.

[106] Jingru Tan, Changbao Wang, Buyu Li, Quanquan Li, Wanli Ouyang, Changqing Yin, and Junjie Yan. Equalization loss for long-tailed object recognition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11662–11671, 2020.

[107] Xianfeng Tang, Huaxiu Yao, Yiwei Sun, Yiqi Wang, Jiliang Tang, Charu Aggarwal, Prasenjit Mitra, and Suhang Wang. Investigating and mitigating degree-related biases in graph convol-

tuional networks. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pages 1435–1444, 2020.

[108] Zhenwei Tang, Shichao Pei, Zhao Zhang, Yongchun Zhu, Fuzhen Zhuang, Robert Hoehndorf, and Xiangliang Zhang. Positive-unlabeled learning with adversarial data augmentation for knowledge graph completion. *arXiv preprint arXiv:2205.00904*, 2022.

[109] Komal Teru, Etienne Denis, and Will Hamilton. Inductive relation prediction by subgraph reasoning. In *International Conference on Machine Learning*, pages 9448–9457. PMLR, 2020.

[110] Sunil Thulasidasan, Gopinath Chennupati, Jeff A Bilmes, Tanmoy Bhattacharya, and Sarah Michalak. On mixup training: Improved calibration and predictive uncertainty for deep neural networks. *Advances in Neural Information Processing Systems*, 32, 2019.

[111] Kristina Toutanova and Danqi Chen. Observed versus latent features for knowledge base and text inference. In *Proceedings of the 3rd workshop on continuous vector space models and their compositionality*, pages 57–66, 2015.

[112] Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. Complex embeddings for simple link prediction. In *International conference on machine learning*, pages 2071–2080. PMLR, 2016.

[113] Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha Talukdar. Composition-based multi-relational graph convolutional networks. In *International Conference on Learning Representations*, 2019.

[114] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[115] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *stat*, 1050:20, 2017.

[116] Yoav Wald, Amir Feder, Daniel Greenfeld, and Uri Shalit. On calibration and out-of-domain generalization. *Advances in neural information processing systems*, 34:2215–2227, 2021.

[117] Haorui Wang, Haoteng Yin, Muhan Zhang, and Pan Li. Equivariant and stable positional encoding for more powerful graph neural networks. *arXiv preprint arXiv:2203.00199*, 2022.

[118] Peifeng Wang, Shuangyin Li, and Rong Pan. Incorporating gan for negative sampling in knowledge representation learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[119] Tao Wang, Yu Li, Bingyi Kang, Junnan Li, Junhao Liew, Sheng Tang, Steven Hoi, and Jiashi

Feng. The devil is in classification: A simple framework for long-tail instance segmentation. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIV 16*, pages 728–744. Springer, 2020.

[120] Xiyuan Wang, Haotong Yang, and Muhan Zhang. Neural common neighbor with completion for link prediction. *arXiv preprint arXiv:2302.00890*, 2023.

[121] Xiyuan Wang, Haotong Yang, and Muhan Zhang. Neural common neighbor with completion for link prediction. *arXiv preprint arXiv:2302.00890*, 2023.

[122] Xudong Wang, Long Lian, Zhongqi Miao, Ziwei Liu, and Stella Yu. Long-tailed recognition by routing diverse distribution-aware experts. In *International Conference on Learning Representations*, 2021.

[123] Zhitao Wang, Yong Zhou, Litao Hong, Yuanhang Zou, Hanjing Su, and Shouzhi Chen. Pairwise learning for neural link prediction. *arXiv preprint arXiv:2112.02936*, 2021.

[124] Qitian Wu, Wentao Zhao, Zenan Li, David P Wipf, and Junchi Yan. Nodeformer: A scalable graph structure learning transformer for node classification. *Advances in Neural Information Processing Systems*, 35:27387–27401, 2022.

[125] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[126] Wenhan Xiong, Thien Hoang, and William Yang Wang. Deeppath: A reinforcement learning method for knowledge graph reasoning. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 564–573, 2017.

[127] Bishan Yang, Scott Wen-tau Yih, Xiaodong He, Jianfeng Gao, and Li Deng. Embedding entities and relations for learning and inference in knowledge bases. In *Proceedings of the International Conference on Learning Representations (ICLR) 2015*, 2015.

[128] Fan Yang, Zhilin Yang, and William W Cohen. Differentiable learning of logical rules for knowledge base reasoning. *Advances in neural information processing systems*, 30, 2017.

[129] Yang Yang, Ryan N Lichtenwalter, and Nitesh V Chawla. Evaluating link prediction methods. *Knowledge and Information Systems*, 45(3):751–782, 2015.

[130] Zhen Yang, Ming Ding, Chang Zhou, Hongxia Yang, Jingren Zhou, and Jie Tang. Understanding negative sampling in graph representation learning. In *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 1666–1676, 2020.

[131] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting semi-supervised learning

with graph embeddings. In *International conference on machine learning*, pages 40–48. PMLR, 2016.

[132] Liang Yao, Chengsheng Mao, and Yuan Luo. Graph convolutional networks for text classification. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 7370–7377, 2019.

[133] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? *Advances in Neural Information Processing Systems*, 34:28877–28888, 2021.

[134] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 974–983, 2018.

[135] Bo Yuan and Xiaoli Ma. Sampling+ reweighting: Boosting the performance of adaboost on imbalanced datasets. In *The 2012 international joint conference on neural networks (IJCNN)*, pages 1–6. IEEE, 2012.

[136] Seongjun Yun, Seoyoon Kim, Junhyun Lee, Jaewoo Kang, and Hyunwoo J Kim. Neo-gnns: Neighborhood overlap-aware graph neural networks for link prediction. *Advances in Neural Information Processing Systems*, 34:13683–13694, 2021.

[137] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations*, 2018.

[138] L Zhang, Z Deng, K Kawaguchi, A Ghorbani, and J Zou. How does mixup help with robustness and generalization? In *International Conference on Learning Representations*, 2021.

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

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

[141] Muhan Zhang, Pan Li, Yinglong Xia, Kai Wang, and Long Jin. Labeling trick: A theory of using graph neural networks for multi-node representation learning. *Advances in Neural Information Processing Systems*, 34:9061–9073, 2021.

[142] Muhan Zhang, Pan Li, Yinglong Xia, Kai Wang, and Long Jin. Labeling trick: A theory of using graph neural networks for multi-node representation learning. *Advances in Neural Information Processing Systems*, 34:9061–9073, 2021.

[143] Yongqi Zhang and Quanming Yao. Knowledge graph reasoning with relational digraph. In *Proceedings of the ACM Web Conference 2022*, pages 912–924, 2022.

[144] Yongqi Zhang, Zhanke Zhou, Quanming Yao, Xiaowen Chu, and Bo Han. Adaprop: Learning adaptive propagation for graph neural network based knowledge graph reasoning. In *KDD*, 2023.

[145] He Zhao, Lan Du, and Wray Buntine. Leveraging node attributes for incomplete relational data. In *International conference on machine learning*, pages 4072–4081. PMLR, 2017.

[146] Tianxiang Zhao, Xiang Zhang, and Suhang Wang. Graphsmote: Imbalanced node classification on graphs with graph neural networks. In *Proceedings of the 14th ACM international conference on web search and data mining*, pages 833–841, 2021.

[147] Tong Zhao, Wei Jin, Yozen Liu, Yingheng Wang, Gang Liu, Stephan Günneman, Neil Shah, and Meng Jiang. Graph data augmentation for graph machine learning: A survey. *arXiv preprint arXiv:2202.08871*, 2022.

[148] Tong Zhao, Gang Liu, Daheng Wang, Wenhao Yu, and Meng Jiang. Counterfactual graph learning for link prediction. *arXiv preprint arXiv:2106.02172*, 2021.

[149] Tong Zhao, Yozen Liu, Leonardo Neves, Oliver Woodford, Meng Jiang, and Neil Shah. Data augmentation for graph neural networks. In *Proceedings of the aaai conference on artificial intelligence*, volume 35, pages 11015–11023, 2021.

[150] Boyan Zhou, Quan Cui, Xiu-Shen Wei, and Zhao-Min Chen. Bbn: Bilateral-branch network with cumulative learning for long-tailed visual recognition. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9719–9728, 2020.

[151] Tao Zhou, Linyuan Lü, and Yi-Cheng Zhang. Predicting missing links via local information. *The European Physical Journal B*, 71:623–630, 2009.

[152] Zhaocheng Zhu, Xinyu Yuan, Louis-Pascal Xhonneux, Ming Zhang, Maxime Gazeau, and Jian Tang. Learning to efficiently propagate for reasoning on knowledge graphs. *arXiv preprint arXiv:2206.04798*, 2022.

[153] Zhaocheng Zhu, Zuobai Zhang, Louis-Pascal Xhonneux, and Jian Tang. Neural bellman-ford networks: A general graph neural network framework for link prediction. *Advances in Neural Information Processing Systems*, 34:29476–29490, 2021.

[154] Zhaocheng Zhu, Zuobai Zhang, Louis-Pascal Xhonneux, and Jian Tang. Neural bellman-ford networks: A general graph neural network framework for link prediction. *Advances in Neural Information Processing Systems*, 34:29476–29490, 2021.

APPENDIX A

EVALUATING GRAPH NEURAL NETWORKS FOR LINK PREDICTION: CURRENT PITFALLS AND NEW BENCHMARKING

A.1 Common Neighbor Distribution

In Figure 2.1 we demonstrate the common neighbor (CN) distribution among positive and negative test samples for ogbl-collab, ogbl-ppa, and ogbl-citation2. These results demonstrate that a vast majority of negative samples have no CNs. Since CNs is a typically good heuristic, this makes it easy to identify most negative samples.

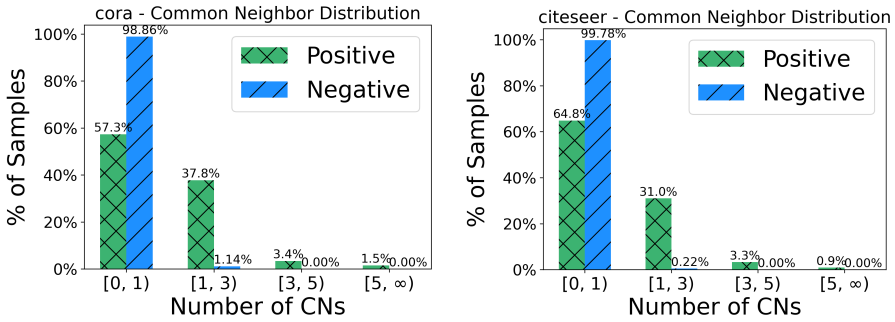
We further present the CN distribution of Cora, Citeseer, Pubmed, and ogbl-ddi in Figure A.1. The CN distribution of Cora, Citeseer, and Pubmed are consistent with our previous observations on the OGB datasets in Figure 2.1. We note that ogbl-ddi exhibits a different distribution with other datasets. As compared to the other datasets, most of the negative samples in ogbl-ddi have common neighbors. This is likely because ogbl-ddi is considerably denser than the other graphs. As shown in Table A.1, the average node degree in ogbl-ddi is 625.68, significantly larger than the second largest dataset ogbl-ppa with 105.25. Thus, despite the random sampling of negative samples, the high degree of node connectivity within the ogbl-ddi graph predisposes a significant likelihood for the occurrence of common neighbors.

We also present the CN distributions under the HeaRT setting. The plots for Cora, Citeseer, Pubmed are shown in Figure A.2. The plots for the OGB datasets are shown in Figure A.3. We observe that the CN distribution of HeaRT is more aligned with the positive samples. This allows for a fairer evaluation setting by not favoring models that use CN information.

A.2 Additional Definitions

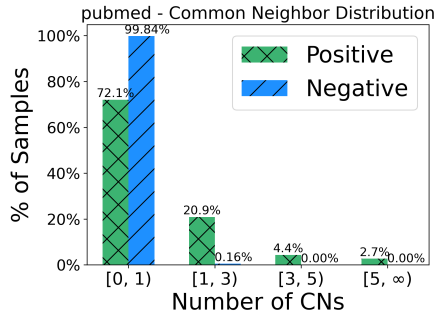
A.2.1 Evaluation Metrics

In this section we define the various evaluation metrics used. Given a single positive sample and M negative samples, we first score each sample and then rank the positive sample among the negatives. The rank is then given by rank_i. I.e., a rank of 1 indicates that the positive sample

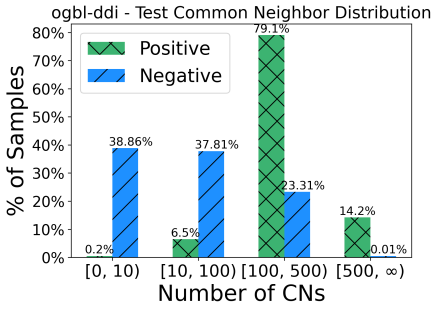


(a) Cora

(b) Citeseer



(c) Pubmed



(d) ogbl-ddi

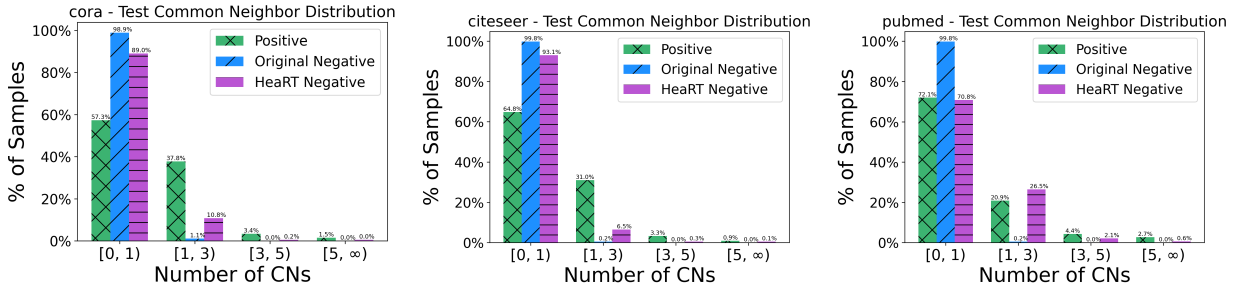
Figure A.1 Common neighbor distribution for the positive and negative test samples for the Cora, Citeseer, Pubmed, and ogbl-ddi under the existing evaluation setting.

has a higher score than all negatives. The hope is that the positive sample ranks above most or all negative samples. Various metrics make use of this rank. We use N to denote the number of positive samples.

Hits@K. It measures whether the true positive is within the top K predictions or not: $\text{Hits@K} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}(\text{rank}_i \leq \text{K})$. rank_i is the rank of the i-th sample. The indicator function $\mathbf{1}$ is 1 if $\text{rank}_i \leq \text{K}$, and 0 otherwise.

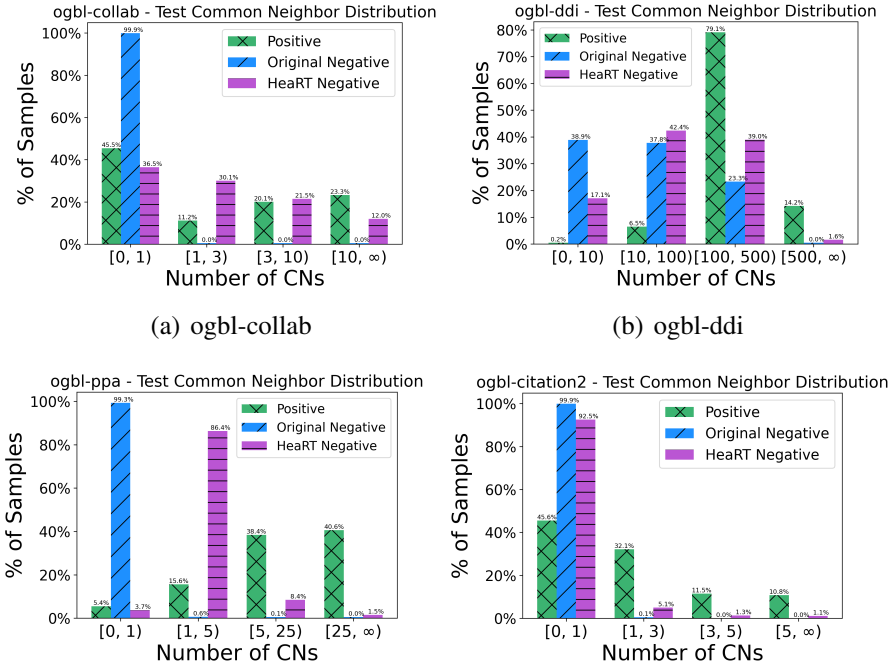
Mean Reciprocal Rank (MRR). It is the mean of the reciprocal rank over all positive samples: $\text{MRR} = \frac{1}{N}\sum_{i=1}^{N} \frac{1}{\text{rank}_i}$, where rank_i is the rank of the i-th sample.

AUC. It measures the likelihood that a positive sample is ranked higher than a random negative sample: $\text{AUC} = \frac{\sum_{i\in\mathcal{D}^0}\sum_{j\in\mathcal{D}^1}\mathbf{1}(\text{rank}_i<\text{rank}_j)}{|\mathcal{D}^0|\cdot|\mathcal{D}^1|}$, where $\mathcal{D}^0$ is the set of positive samples, $\mathcal{D}^1$ is the set of negative samples, and rank_i is the rank of the i-th sample. The indicator function $\mathbf{1}$ is 1 if $\text{rank}_i < \text{rank}_j$, and 0 otherwise.



(a) Cora (b) Citeseer (c) Pubmed

Figure A.2 Common neighbor distribution for the positive negative samples under both evaluation settings for Cora, Citeseer, Pubmed.



(a) ogbl-collab (b) ogbl-ddi

(c) ogbl-ppa (d) ogbl-citation

Figure A.3 Common neighbor distribution for the positive negative samples under both evaluation settings for the OGB datasets.

A.2.2 Negative Sampling

Since only positive links are observed, there is a need to generate negative links (i.e., edges that don't exist in $\mathcal{G}$) to both train and evaluate different models. We detail how these samples are generated in both training and evaluation.

Training Negative Samples. During training, the negative samples are randomly selected, with all nodes being equally likely to be selected. Let $\mathcal{V}$ and $\mathcal{E}$ be the set of nodes and edges in $\mathcal{G}$. Furthermore, we define $v \in \text{Rand}(\mathcal{V})$ as returning a random node in $\mathcal{V}$. A single negative sample (a^-, b^-) is given by:

$$(a^-, b^-) = (\text{Rand}(\mathcal{V}), \text{Rand}(\mathcal{V})) \,. \tag{A.1}$$

Typically one negative sample is generated per positive sample.

Evaluation Negative Samples. For the existing setting, a fixed set of randomly selected samples are used as negatives during evaluation. Furthermore, the same set of negative samples are used for each positive sample. This is equivalent to Eq. (A.1). The only exception is the ogbl-citation2 [43] dataset. For ogbl-citation2, each positive sample is only evaluated against its own set 1000 negative samples. For a positive sample, its negative samples are restricted to contain one of its two nodes (i.e., a corruption). The other node is randomly selected from $\mathcal{V}$. This is equivalent to selecting a set of random samples from the set $S(a, b)$ as defined in Eq. (2.3).

A.3 Datasets and Experimental Settings

A.3.1 Datasets

Table A.1 Statistics of datasets. The split ratio is for train/validation/test.

	Cora	Citeseer	Pubmed	ogbl-collab	ogbl-ddi	ogbl-ppa	ogbl-citation2
#Nodes	2,708	3,327	18,717	235,868	4,267	576,289	2,927,963
#Edges	5,278	4,676	44,327	1,285,465	1,334,889	30,326,273	30,561,187
Mean Degree	3.9	2.81	4.74	10.90	625.68	105.25	20.88
Split Ratio	85/5/10	85/5/10	85/5/10	92/4/4	80/10/10	70/20/10	98/1/1

The statistics of datasets are shown in Table A.1. Generally, Cora, Citeseer, and Pubmed are smaller graphs, with the OGB datasets having more nodes and edges. We adopt the single fixed train/validation/test split with percentages 85/5/10% for Cora, Citeseer, and Pubmed. For OGB datasets, we use the fixed splits provided by the OGB benchmark [43].

A.3.2 Experimental Settings

Training Settings. We use the binary cross entropy loss to train each model. The loss is optimized using the Adam optimizer [52]. During training we randomly sample one negative

sample per positive sample. Each model is trained for a maximum of 9999 epochs, with the process set to terminate when there are no improvements observed in the validation performance over n checkpoints. The choice of n is influenced by both the specific dataset and the complexity of the model. For smaller datasets, such as Cora, Citeseer, and Pubmed, we set $n = 50$ uniformly across models (except for NBFNet and SEAL where $n = 20$ due to their computational inefficiency). When training on larger OGB datasets, we use a stratified approach: $n = 100$ for the simpler methods (i.e., the embedding and GNN-based models) and $n = 20$ for the more advanced methods. This is due to the increased complexity and runtime of more advanced methods. An exception is for ogbl-citation2, the largest dataset. To accommodate for its size, we limit the maximum number of epochs to the recommended value from each model's source code. Furthermore, we set $n = 20$ for all models.

In order to accommodate the computational requirements for our extensive experiments, we harness a variety of high-capacity GPU resources. This includes: Tesla V100 32Gb, NVIDIA RTX A6000 48Gb, NVIDIA RTX A5000 24Gb, and Quadro RTX 8000 48Gb.

Table A.2 Hyperparameter Search Ranges.

Dataset	Learning Rate	Dropout	Weight Decay	# Model Layers	# Prediction Layers	Embedding Dim
Cora	(0.01, 0.001)	(0.1, 0.3, 0.5)	(1e-4, 1e-7, 0)	(1, 2, 3)	(1, 2, 3)	(128, 256)
Citeseer	(0.01, 0.001)	(0.1, 0.3, 0.5)	(1e-4, 1e-7, 0)	(1, 2, 3)	(1, 2, 3)	(128, 256)
Pubmed	(0.01, 0.001)	(0.1, 0.3, 0.5)	(1e-4, 1e-7, 0)	(1, 2, 3)	(1, 2, 3)	(128, 256)
ogbl-collab	(0.01, 0.001)	(0, 0.3, 0.5)	0	3	3	256
ogbl-ddi	(0.01, 0.001)	(0, 0.3, 0.5)	0	3	3	256
ogbl-ppa	(0.01, 0.001)	(0, 0.3, 0.5)	0	3	3	256
ogbl-citation2	(0.01, 0.001)	(0, 0.3, 0.5)	0	3	3	128

Hyperparameter Settings. We present the hyparameter searching range in Table A.2. For the smaller graphs, Cora, Citeseer, and Pubmed, we have a larger search space. However, it's not feasible to tune over such large space for OGB datasets. By following the most commonly used settings among published hyperparameters, we fix the weight decay, number of model and prediction layers, and the embedding dimension. Furthermore, due to GPU memory constraints, the embedding size is reduced to be 128 for the largest dataset ogbl-citation2.

We note that several exceptions exist to these ranges when they result in significant performance

degradations. In such instances, adjustments are guided by the optimal hyperparameters published in the respective source codes. This includes:

- PEG [117]: Adhering to the optimal hyperparameters presented in the source code,[1] when training on ogbl-ddi we set the number of model layers to 2 and the maximum number of epochs to 400.
- NCN/NCNC [120]: When training on ogbl-ddi, we adhere to the suggested optimal hyperparameters used in the source code.[2] Specifically, we set the number of model layers to be 1, and we don't apply the pretraining for NCNC to facilitate a fair comparison.
- NBFNet [153]: Due to the expensive nature of NBFNet, we further fix the weight decay to 0 when training on Cora, Citeseer, and Pubmed. Furthermore, we follow the suggested hyperparameters [3] and set the embedding dimension to be 32 and the number of model layers to be 6.
- SEAL [139]: Due to the computational inefficiency of SEAL, when training on Cora, Citeseer and Pubmed we further fix the weight decay to 0. Furthermore, we adhere to the published hyperparameters [4] and fix the number of model layers to be 3 and the embedding dimension to be 256.
- BUDDY [20]: When training on ogbl-ppa, we incorporate the RA and normalized degree as input features while excluding the raw node features. This is based on the optimal hyperparameters published by the authors.[5]

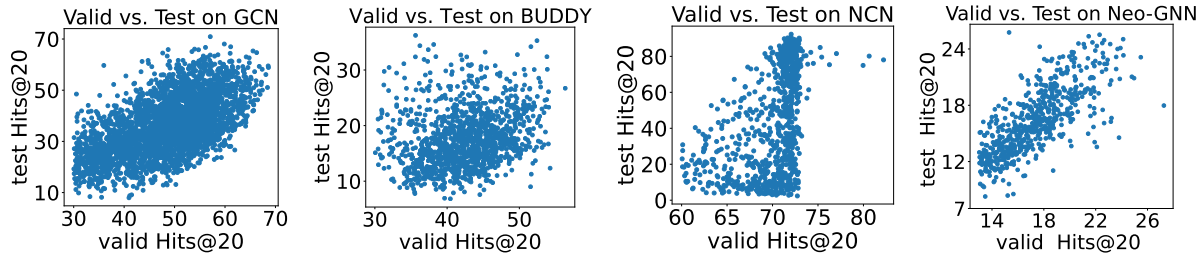
A.4 Reported vs. Our Results on ogbl-ddi

In Section 2.3 (see observation 2), we remarked that there is divergence between the reported results and our results on ogbl-ddi for some methods. A comprehensive comparison of this discrepancy is shown in Table A.3. The reported results for Node2Vec, MF, GCN, and SAGE are taken from [43]. The results for the other methods are from their original paper: SEAL [141], BUDDY [20], Neo-GNN [136], NCN [120], NCNC [120], and PEG [117].

[1] https://github.com/Graph-COM/PEG/
[2] https://github.com/GraphPKU/NeuralCommonNeighbor/
[3] https://github.com/DeepGraphLearning/NBFNet/
[4] https://github.com/facebookresearch/SEAL_OGB/
[5] https://github.com/melifluos/subgraph-sketching/

Table A.3 Comparison results between ours and reported results on ogbl-ddi (Hits@20).

	Node2Vec	MF	GCN	SAGE	SEAL	BUDDY	Neo-GNN	NCN	NCNC	PEG
Reported	23.26 ± 2.09	13.68 ± 4.75	37.07 ± 5.07	**53.90 ± 4.74**	**30.56 ± 3.86**	**78.51 ± 1.36**	**63.57 ± 3.52**	**82.32 ± 6.10**	**84.11 ± 3.67**	**43.80 ± 0.32**
Ours	**34.69 ± 2.90**	**23.50 ± 5.35**	**49.90 ± 7.23**	49.84 ± 15.56	25.25 ± 3.90	29.60 ± 4.75	20.95 ± 6.03	76.52 ± 10.47	70.23 ± 12.11	30.28 ± 4.92



(a) GCN (b) BUDDY (c) NCN (d) Neo-GNN

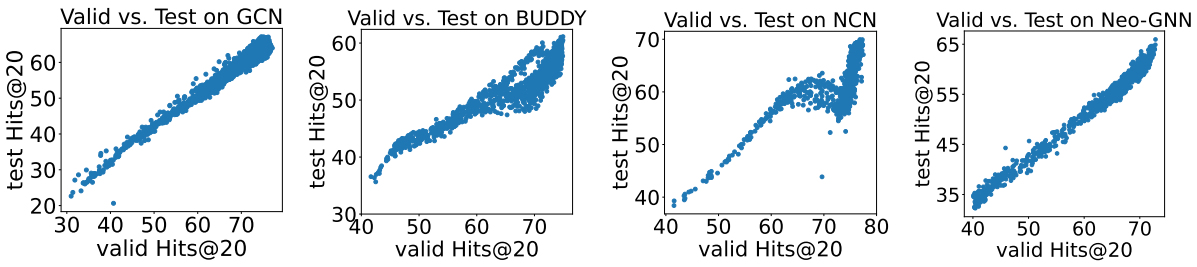
Figure A.4 Validation vs. test performance for GCN, BUDDY, NCN and Neo-GNN on ogbl-ddi under the existing evaluation setting.

A.5 Additional Investigation on ogbl-ddi

In this section, we present the additional investigation on the ogbl-ddi dataset. In Section A.5.1 we examine under the existing evaluation setting, there exists a poor relationship between the validation and test performance on ogbl-ddi for many methods. We then demonstrate in Section A.5.2 that under HeaRT this problem is lessened.

A.5.1 Existing Evaluation Setting

Upon inspection, we found that there is a poor relationship between the validation and test performance on ogbl-ddi. Since we choose the best hyperparameters based on the validation set, this makes it difficult to properly tune any model on ogbl-ddi. To demonstrate this point, we record the validation and test performance at multiple checkpoints during the training process. The experiments are conducted over 10 seeds. To ensure that our results are not caused by our hyperparameter settings, we use the reported hyperparameters for each model. Lastly, we plot the results for GCN, BUDDY, NCN, and Neo-GNN in Figure A.4. It is clear from the results that there exists a poor relationship between the validation and test performance. For example, for NCN, a validation performance of 70 can imply a test performance of 3 to 80. Further investigation is needed to uncover the cause of this misalignment.



(a) GCN (b) BUDDY (c) NCN (d) Neo-GNN

Figure A.5 Validation vs. test performance when utilizing HeaRT for GCN, BUDDY, NCN and Neo-GNN on ogbl-ddi. We find they have a much stronger relationship than under the existing setting (see Figure A.4).

A.5.2 New Evaluation Setting

Under our new setting, we find that the validation and test performance have a much better relationship. In Section A.5.1 we observed that there exists a poor relationship between the validation and test performance on ogbl-ddi under the existing evaluation setting. This meddles with our ability to choose the best hyperparameters for each model, as good validation performance is not indicative of good test performance. However, this does not seem to be the case under the new evaluation setting. In Figure A.5 we plot the relationship between the validation and test performance by checkpoint for various models. Compared to the same plots under the existing setting (Figure A.4), the new results display a much better relationship.

While it's unclear what is the cause of the poor relationship between the test and validation performance under the existing setting, we conjecture that tailoring the negatives to each positive sample allows for a more natural comparison between a positive sample and its negatives. This may help produce more stable evaluation metrics, thereby strengthening the alignment between the validation and test performance.

A.6 Additional Results Under the Existing Setting

We present additional results of Cora, Citeseer, Pubmed and OGB datasets in Tables A.4-A.7 under the existing setting. We also omit the MRR for ogbl-collab, ogbl-ddi, and ogbl-ppa. This is because the large number of negative samples make it very inefficient to calculate.

Table A.4 Additional results on Cora(%) under the existing evaluation setting. We highlight the results ranked first, second, and third as green, blue, and orange, respectively.

	Models	Hits@1	Hits@3	Hits@10	Hits@100
Heuristic	CN	13.47	13.47	42.69	42.69
	AA	22.2	39.47	42.69	42.69
	RA	20.11	39.47	42.69	42.69
	Shortest Path	0	0	42.69	71.35
	Katz	19.17	28.46	51.61	74.57
Embedding	Node2Vec	**22.3 ± 11.76**	**41.63 ± 10.5**	62.34 ± 2.35	84.88 ± 0.96
	MF	7.76 ± 5.61	13.26 ± 4.52	29.16 ± 6.68	66.39 ± 5.03
	MLP	18.79 ± 11.40	35.35 ± 10.71	53.59 ± 3.57	85.52 ± 1.44
GNN	GCN	16.13 ± 11.18	32.54 ± 10.83	66.11 ± 4.03	91.29 ± 1.25
	GAT	18.02 ± 8.96	**42.28 ± 6.37**	63.82 ± 2.72	90.70 ± 1.03
	SAGE	**29.01 ± 6.42**	**44.51 ± 6.57**	63.66 ± 4.98	91.00 ± 1.52
	GAE	17.57 ± 4.37	24.82 ± 4.91	**70.29 ± 2.75**	**92.75 ± 0.95**
GNN+heuristic	SEAL	12.35 ± 8.57	38.63 ± 4.96	55.5 ± 3.28	84.76 ± 1.6
	BUDDY	12.62 ± 6.69	29.64 ± 5.71	59.47 ± 5.49	91.42 ± 1.26
	Neo-GNN	4.53 ± 1.96	33.36 ± 9.9	64.1 ± 4.31	87.76 ± 1.37
	NCN	19.34 ± 9.02	38.39 ± 7.01	**74.38 ± 3.15**	**95.56 ± 0.79**
	NCNC	9.79 ± 4.56	34.31 ± 8.87	**75.07 ± 1.95**	**95.62 ± 0.84**
	NBFNet	**29.94 ± 5.78**	38.29 ± 3.03	62.79 ± 2.53	88.63 ± 0.46
	PEG	5.88 ± 1.65	30.53 ± 6.42	62.49 ± 4.05	91.42 ± 0.8

Table A.5 Additional results on Citeseer(%) under the existing evaluation setting. We highlight the results ranked first, second, and third as green, blue, and orange, respectively.

	Models	Hits@1	Hits@3	Hits@10	Hits@100
Heuristic	CN	13.85	35.16	35.16	35.16
	AA	21.98	35.16	35.16	35.16
	RA	18.46	35.16	35.16	35.16
	Shortest Path	0	53.41	56.92	62.64
	Katz	24.18	54.95	57.36	62.64
Embedding	Node2Vec	30.24 ± 16.37	54.15 ± 6.96	68.79 ± 3.05	89.89 ± 1.48
	MF	19.25 ± 6.71	29.03 ± 4.82	38.99 ± 3.26	59.47 ± 2.69
	MLP	30.22 ± 10.78	56.42 ± 7.90	69.74 ± 2.19	91.25 ± 1.90
GNN	GCN	37.47 ± 11.30	62.77 ± 6.61	74.15 ± 1.70	91.74 ± 1.24
	GAT	34.00 ± 11.14	62.72 ± 4.60	74.99 ± 1.78	91.69 ± 2.11
	SAGE	27.08 ± 10.27	65.52 ± 4.29	78.06 ± 2.26	**96.50 ± 0.53**
	GAE	**54.06 ± 5.8**	65.3 ± 2.54	**81.72 ± 2.62**	95.17 ± 0.5
GNN+heuristic	SEAL	31.25 ± 8.11	46.04 ± 5.69	60.02 ± 2.34	85.6 ± 2.71
	BUDDY	**49.01 ± 15.07**	**67.01 ± 6.22**	**80.04 ± 2.27**	95.4 ± 0.63
	Neo-GNN	41.01 ± 12.47	59.87 ± 6.33	69.25 ± 1.9	89.1 ± 0.97
	NCN	35.52 ± 13.96	66.83 ± 4.06	79.12 ± 1.73	**96.17 ± 1.06**
	NCNC	**53.21 ± 7.79**	**69.65 ± 3.19**	**82.64 ± 1.4**	**97.54 ± 0.59**
	NBFNet	17.25 ± 5.47	51.87 ± 2.09	68.97 ± 0.77	86.68 ± 0.42
	PEG	39.19 ± 8.31	**70.15 ± 4.3**	77.06 ± 3.53	94.82 ± 0.81

Table A.6 Additional results on Pubmed(%) under the existing evaluation setting. We highlight the results ranked first, second, and third as green, blue, and orange, respectively.

	Models	Hits@1	Hits@3	Hits@10	Hits@100
Heuristic	CN	7.06	12.95	27.93	27.93
	AA	12.95	16	27.93	27.93
	RA	11.67	15.21	27.93	27.93
	Shortest Path	0	0	27.93	60.36
	Katz	12.88	25.38	42.17	61.8
Embedding	Node2Vec	**29.76 ± 4.05**	34.08 ± 2.43	44.29 ± 2.62	63.07 ± 0.34
	MF	12.58 ± 6.08	22.51 ± 5.6	32.05 ± 2.44	53.75 ± 2.06
	MLP	7.83 ± 6.40	17.23 ± 2.79	34.01 ± 4.94	84.19 ± 1.33
GNN	GCN	5.72 ± 4.28	19.82 ± 7.59	56.06 ± 4.83	87.41 ± 0.65
	GAT	6.45 ± 10.37	23.02 ± 10.49	46.77 ± 4.03	80.95 ± 0.72
	SAGE	11.26 ± 6.86	27.23 ± 7.48	48.18 ± 4.60	**90.02 ± 0.70**
	GAE	1.99 ± 0.12	31.75 ± 1.13	45.48 ± 1.07	84.3 ± 0.31
GNN+heuristic	SEAL	**30.93 ± 8.35**	**40.58 ± 6.79**	48.45 ± 2.67	76.06 ± 4.12
	BUDDY	15.31 ± 6.13	29.79 ± 6.76	46.62 ± 4.58	83.21 ± 0.59
	Neo-GNN	19.95 ± 5.86	34.85 ± 4.43	**56.25 ± 3.42**	86.12 ± 1.18
	NCN	26.38 ± 6.54	**36.82 ± 6.56**	**62.15 ± 2.69**	**90.43 ± 0.64**
	NCNC	9.14 ± 5.76	33.01 ± 6.28	**61.89 ± 3.54**	**91.93 ± 0.6**
	NBFNet	**40.47 ± 2.91**	**44.7 ± 2.58**	54.51 ± 0.84	79.18 ± 0.71
	PEG	8.52 ± 3.73	24.46 ± 6.94	45.11 ± 4.02	76.45 ± 3.83

Table A.7 Additional results on OGB datasets(%) under the existing evaluation setting. We highlight the results ranked first, second, and third as green, blue, and orange, respectively.

	ogbl-collab		ogbl-ddi		ogbl-ppa		ogbl-citation2		
	Hits@20	Hits@100	Hits@50	Hits@100	Hits@20	Hits@50	Hits@20	Hits@50	Hits@100
CN	49.98	65.6	26.51	34.52	13.26	19.67	77.99	77.99	77.99
AA	55.79	65.6	27.07	36.35	14.96	21.83	77.99	77.99	77.99
RA	55.01	65.6	19.14	31.17	25.64	**38.81**	77.99	77.99	77.99
Shortest Path	46.49	66.82	0	0	0	0	>24h	>24h	>24h
Katz	**58.11**	**71.04**	26.51	34.52	13.26	19.67	78	78	78
Node2Vec	40.68 ± 1.75	55.58 ± 0.77	59.19 ± 3.61	73.49 ± 3.18	11.22 ± 1.91	19.22 ± 1.69	82.8 ± 0.13	92.33 ± 0.1	96.44 ± 0.03
MF	39.99 ± 1.25	43.22 ± 1.94	45.51 ± 11.13	61.72 ± 6.56	9.33 ± 2.83	21.08 ± 3.92	70.8 ± 12.0	74.48 ± 10.42	75.5 ± 10.13
MLP	27.66 ± 1.61	42.13 ± 1.09	N/A	N/A	0.16 ± 0.0	0.26 ± 0.03	74.16 ± 0.1	86.59 ± 0.08	93.14 ± 0.06
GCN	44.92 ± 3.72	62.67 ± 2.14	74.54 ± 4.74	85.03 ± 3.41	11.17 ± 2.93	21.04 ± 3.11	**98.01 ± 0.04**	**99.03 ± 0.02**	**99.48 ± 0.02**
GAT	43.59 ± 4.17	62.24 ± 2.29	55.46 ± 10.16	69.74 ± 10.01	OOM	OOM	OOM	OOM	OOM
SAGE	50.77 ± 2.33	65.36 ± 1.05	**93.48 ± 1.36**	**97.37 ± 0.55**	19.37 ± 2.65	31.3 ± 2.36	97.48 ± 0.03	98.75 ± 0.03	99.3 ± 0.02
GAE	OOM	OOM	12.39 ± 8.74	14.03 ± 9.22	OOM	OOM	OOM	OOM	OOM
SEAL	54.19 ± 1.57	69.94 ± 0.72	43.34 ± 3.23	52.2 ± 1.78	21.81 ± 4.3	36.88 ± 4.06	94.61 ± 0.11	95.0 ± 0.12	95.37 ± 0.14
BUDDY	**57.78 ± 0.59**	67.87 ± 0.87	53.36 ± 2.57	71.04 ± 2.56	**26.33 ± 2.63**	38.18 ± 1.32	**97.79 ± 0.07**	**98.86 ± 0.04**	99.38 ± 0.03
Neo-GNN	**57.05 ± 1.56**	**71.76 ± 0.55**	33.88 ± 10.1	46.55 ± 13.29	26.16 ± 1.24	37.95 ± 1.45	97.05 ± 0.07	98.75 ± 0.03	**99.41 ± 0.02**
NCN	50.27 ± 2.72	67.58 ± 0.09	**95.51 ± 0.87**	**97.54 ± 0.7**	**40.29 ± 2.22**	**53.35 ± 1.77**	**97.97 ± 0.03**	**99.02 ± 0.02**	**99.5 ± 0.01**
NCNC	54.91 ± 2.84	**70.91 ± 0.25**	**92.34 ± 2.42**	**96.35 ± 0.52**	**40.1 ± 1.06**	**52.09 ± 1.99**	97.22 ± 0.78	98.2 ± 0.71	98.77 ± 0.6
NBFNet	OOM	OOM	>24h	>24h	OOM	OOM	OOM	OOM	OOM
PEG	33.57 ± 7.40	55.14 ± 2.10	47.93 ± 3.18	59.95 ± 2.52	OOM	OOM	OOM	OOM	OOM

A.7 Additional Details on HeaRT

As described in Section 2.4.2, given a positive sample (a, b), we seeks to generate K negative samples to evaluate against. The negative samples are drawn from the set of possible corruptions of (a, b), i.e, $S(a, b)$ (see Eq. (2.3)). Multiple heuristics are used to determine which K negative samples to use. Furthermore, the negative samples are split evenly between both nodes. That is, we generate $K/2$ negative samples that contain either node a and b, respectively. This process is illustrated in Figure 2.2.

The rest of this section is structured as follows. In Section A.7.1 we describe how we use multiple heuristics for estimating the difficulty of negative samples. Then in Section A.7.2 we describe how we combine the ranks given by different heuristic methods.

A.7.1 Determining Hard Negative Samples

We are first tasked with *how to choose the negative samples*. As discussed and shown in Section 2.4.2, we want to select the negative samples from $S(a, b)$ such that they are non-trivial to classify. Hence, as inspired by the candidate generation process in real-world recommender systems [41, 33], we aim to select a set of 'hard' negative samples that are more relevant to the source node. The candidate generation process is typically based on some primitive and simple link prediction heuristics. These heuristics can be also treated as link prediction methods (see Tables 2.1 and 2.2).

We use multiple heuristics that capture a variety of different information. Most link prediction heuristics can be categorized into two main categories: local heuristics and global heuristics [69]. Local heuristics attempt to capture the local neighborhood information that exists near the node pair while global heuristics attempt to use the whole graph structure. To capture the *local* information we use resource allocation (RA) [151], a CN-based approach. Existing results show that RA can achieve strong performance on most datasets (see Tables 2.1 and 2.2). To measure the *global* information we use the personalized pagerank score (PPR) [15]. Random walk based methods are commonly used for candidate generation [41, 33]. Lastly, we further include the cosine feature similarity for the Cora, Citeseer, and Pubmed datasets. This is due to the strong performance of

a MLP on those datasets. By combining these heuristics, we are able to generate a diverse set of negative samples for each positive sample.

For each heuristic we then rank all the possible negative samples. We first denote the score of a heuristic i for a pair of nodes a and b as $h_i(a, b)$. Let's say we want to rank all negative samples that contain a node a, i.e., $(a, *)$. The rank across all nodes is given by:

$$R_i = \underset{v \in \bar{V}}{\text{ArgSort}}\ h_i(a, v), \tag{A.2}$$

where R_i denotes the ranking for heuristic i and and $\bar{V}$ is a subset of the set of nodes in the graph V. We apply a filtering process to exclude all positive training samples, self-loops, and the sample itself under consideration from being selected as negative samples. Additionally, when choosing negative samples for the test samples, we disallow validation samples to be chosen as well. As such, we only consider a subset of nodes $\bar{V} \in V$. This is analogous to the filtered setting used in KGC [13]. However, we adopt a distinct filtering strategy for the ogbl-collab dataset, which is a dynamic collaboration graph. Specifically, **positive training samples are not excluded in the generation of negative samples for validation and test**. Similarly, positive validation samples are not omitted when creating negative test samples. This approach is justified for ogbl-collab which is a dynamic graph, as prior collaboration between authors does not necessarily indicates future collaborations. Further details and discussions are provided in Appendix A.9.

We now have all possible negative samples ranked according to multiple heuristics. However, it is unclear how to choose the negative samples from multiple ranked lists. In the next subsection we detail how we combine the ranks according to each heuristic. This will give us a final ranking, of which we can choose the top $K/2$ as the negative samples for that node.

A.7.2 Combining Heuristic Ranks

In this subsection we tackle the problem of combing the negative sample ranks given by multiple heuristics. More concretely, say we use m heuristics and rank all the samples according to each. We want to arrive at a combined ranking R_{total} that is composed of each rank,

$$R_{\text{total}} = \phi(R_1, R_2, \cdots, R_m). \tag{A.3}$$

Algorithm A.1 Generating Negative Samples of Form $(a, *)$

Require:
a = Node to generate samples for
$\bar{V}$ = Possible nodes to use for negative samples
$\mathcal{H} = \{h_1, h_2, \cdots, h_m\}$ ▷ Set of m heuristics

for $i \in |\mathcal{H}|$ **do**
  $R_i = \underset{v \in \bar{V}}{\text{ArgSort}}\ h_i(a, v)$ ▷ Sort by each heuristic individually
end for
for $v \in \bar{V}$ **do**
  $R_{\text{total}}(a, v) = \min\left(R_1(a, v), R_2(a, v), \cdots, R_m(a, v)\right)$ ▷ Combine the rankings
end for

$R_f = \underset{v \in \bar{V}}{\text{ArgSort}}\ R_{\text{total}}(a, v)$ ▷ Sort by combined ranking
return $R_f[:K/2]$ ▷ Return the top K/2 ranked nodes

We model ϕ via Borda's method [12]. Let $R_i(a, v)$ be the rank of the node pair (a, v) for heuristic i. The combined rank $R_{\text{total}}(a, v)$ across m ranked lists is given by:

$$R_{\text{total}}(a, v) = g\left(R_1(a, v), R_2(a, v), \cdots, R_m(a, v)\right), \tag{A.4}$$

where g is an aggregation function. We set $g = \min(\cdot)$. This is done as it allows us to capture a more distinct set of samples by selecting the "best" for each heuristic. This is especially true when there is strong disagreement between the different heuristics. A final ranking is then done on R_{total} to select the top nodes,

$$R_f = \underset{v \in \bar{V}}{\text{ArgSort}}\ R_{\text{total}}, \tag{A.5}$$

where R_f is the final ranking. The highest $K/2$ nodes are then selected from R_f. Lastly, we note that for some nodes there doesn't exist sufficient scores to rank $K/2$ total nodes. In this case the remaining nodes are chosen randomly. The full generation process for a node a is detailed in Algorithm A.1.

A.8 Additional Results Under HeaRT

We present additional results of Cora, Citeseer, Pubmed, and OGB datasets under HeaRT in Table A.8 and Table A.9. These results include other hit metrics not found in the main tables.

Table A.8 Additional results on Cora, Citeseer, and Pubmed(%) under HeaRT. We highlight the results ranked first, second, and third as green, blue, and orange, respectively.

Models	Cora			Citeseer			Pubmed		
	Hits1	Hits3	Hits100	Hits1	Hits3	Hits100	Hits1	Hits3	Hits100
CN	3.98	10.25	38.71	2.2	9.45	33.63	0.47	1.49	19.29
AA	5.31	12.71	38.9	3.96	12.09	33.85	0.74	1.87	20.37
RA	5.31	12.52	38.52	4.18	11.21	34.07	0.72	1.78	20.04
Shortest Path	0.57	2.85	55.6	0.22	3.52	53.85	0	0.02	21.57
Katz	4.64	11.95	59.96	3.74	11.87	55.82	0.74	2.12	32.78
Node2Vec	5.69 ± 0.81	15.1 ± 0.99	77.21 ± 2.34	9.63 ± 0.82	23.5 ± 1.37	84.46 ± 1.86	0.75 ± 0.14	2.4 ± 0.32	52.27 ± 0.65
MF	1.46 ± 0.8	5.46 ± 1.67	59.68 ± 3.41	2.68 ± 0.92	7.25 ± 0.98	53.25 ± 2.91	1.13 ± 0.24	3.25 ± 0.44	50.56 ± 1.11
MLP	5.48 ± 0.99	14.15 ± 1.56	77.0 ± 1.02	10.44 ± 0.82	26.46 ± 1.24	86.83 ± 1.36	1.28 ± 0.22	4.33 ± 0.28	76.34 ± 0.79
GCN	**7.59 ± 0.61**	**17.46 ± 0.82**	**85.47 ± 0.52**	9.27 ± 0.99	23.19 ± 0.98	89.1 ± 2.13	2.09 ± 0.31	5.58 ± 0.27	73.59 ± 0.53
GAT	5.03 ± 0.81	13.66 ± 0.67	80.87 ± 1.32	8.02 ± 1.21	20.09 ± 0.82	86.83 ± 1.09	1.14 ± 0.16	3.06 ± 0.36	67.06 ± 0.69
SAGE	5.48 ± 0.97	**15.43 ± 1.07**	81.61 ± 0.96	8.37 ± 1.62	23.74 ± 1.62	**92.33 ± 0.68**	**3.03 ± 0.46**	**8.19 ± 1.0**	**79.47 ± 0.53**
GAE	**9.72 ± 0.73**	**19.24 ± 0.76**	79.66 ± 0.95	**13.81 ± 0.82**	**27.71 ± 1.34**	85.49 ± 1.37	1.48 ± 0.23	4.05 ± 0.39	59.79 ± 0.67
SEAL	3.89 ± 2.04	10.82 ± 4.04	61.9 ± 13.97	5.08 ± 1.31	13.68 ± 1.32	68.94 ± 2.3	1.47 ± 0.32	4.71 ± 0.68	65.81 ± 2.43
BUDDY	5.88 ± 0.60	13.76 ± 1.03	82.46 ± 1.79	10.09 ± 0.50	26.11 ± 1.26	**92.66 ± 0.92**	**2.24 ± 0.17**	5.93 ± 0.21	72.01 ± 0.46
Neo-GNN	5.71 ± 0.41	13.89 ± 0.82	80.28 ± 1.08	6.81 ± 0.73	17.8 ± 1.19	85.51 ± 1.01	1.90 ± 0.24	**6.07 ± 0.47**	**76.57 ± 0.58**
NCN	4.85 ± 0.81	14.46 ± 0.98	**84.14 ± 1.24**	**16.77 ± 2.05**	**30.51 ± 0.97**	90.42 ± 0.98	1.13 ± 0.18	3.95 ± 0.24	71.46 ± 0.97
NCNC	4.78 ± 0.71	14.72 ± 1.24	**85.62 ± 0.83**	**11.14 ± 0.82**	**27.21 ± 0.96**	**92.73 ± 1.16**	**2.73 ± 0.49**	**7.05 ± 0.72**	**79.22 ± 0.96**
NBFNet	5.31 ± 1.16	14.95 ± 0.72	76.24 ± 0.68	5.95 ± 1.06	14.53 ± 1.19	72.66 ± 0.95	>24h	>24h	>24h
PEG	**6.98 ± 0.57**	14.93 ± 0.61	82.52 ± 1.28	9.93 ± 0.6	21.91 ± 0.59	90.15 ± 1.43	0.88 ± 0.18	2.61 ± 0.39	64.95 ± 1.81

Table A.9 Additional results on OGB datasets(%) under HeaRT. We highlight the results ranked first, second, and third as green, blue, and orange, respectively.

Models	ogbl-collab		ogbl-ddi		ogbl-ppa		ogbl-citation2	
	Hits50	Hits100	Hits50	Hits100	Hits50	Hits100	Hits50	Hits100
CN	30.52	42.80	70.12	86.53	80.53	86.51	57.56	68.04
AA	33.74	45.20	71.08	87.36	81.93	87.55	58.87	69.39
RA	**36.68**	46.42	76.39	90.96	81.65	86.84	58.88	68.83
Shortest Path	33.77	45.85	0	0	1.34	1.4	>24h	>24h
Katz	**39.18**	48.80	70.12	86.53	80.53	86.51	54.97	67.56
Node2Vec	28.56 ± 0.17	41.84 ± 0.25	**98.38 ± 0.7**	99.91 ± 0.01	69.94 ± 0.06	81.88 ± 0.06	61.22 ± 0.16	77.11 ± 0.13
MF	30.83 ± 0.22	43.23 ± 0.34	95.52 ± 0.72	99.54 ± 0.08	83.29 ± 3.35	89.75 ± 1.9	29.64 ± 7.3	65.87 ± 8.37
MLP	28.88 ± 0.32	46.83 ± 0.33	N/A	N/A	5.36 ± 0.0	22.01 ± 0.01	61.29 ± 0.07	76.94 ± 0.1
GCN	35.29 ± 0.49	**50.83 ± 0.21**	97.65 ± 0.68	99.85 ± 0.06	81.48 ± 0.48	89.62 ± 0.23	70.77 ± 0.34	**85.43 ± 0.18**
GAT	32.92 ± 1.41	46.71 ± 0.84	98.15 ± 0.24	99.93 ± 0.02	OOM	OOM	OOM	OOM
SAGE	33.48 ± 1.40	48.33 ± 0.49	**99.17 ± 0.11**	**99.98 ± 0.01**	81.84 ± 0.24	89.46 ± 0.13	**71.91 ± 0.1**	**85.86 ± 0.09**
GAE	OOM	OOM	28.29 ± 13.65	48.34 ± 15.0	OOM	OOM	OOM	OOM
SEAL	33.57 ± 0.84	43.06 ± 1.09	82.42 ± 3.37	92.63 ± 2.05	**87.34 ± 0.49**	**92.45 ± 0.26**	65.11 ± 2.33	77.64 ± 2.43
BUDDY	**39.04 ± 0.11**	**50.49 ± 0.09**	97.81 ± 0.31	**99.93 ± 0.01**	82.5 ± 0.51	88.36 ± 0.32	67.47 ± 0.32	81.94 ± 0.26
Neo-GNN	36.11 ± 2.36	**49.25 ± 0.81**	83.45 ± 11.03	94.7 ± 4.82	81.21 ± 1.39	88.31 ± 0.19	62.14 ± 0.51	79.13 ± 0.42
NCN	34.53 ± 0.98	45.69 ± 0.42	**98.43 ± 0.22**	**99.96 ± 0.01**	**89.37 ± 0.28**	**93.11 ± 0.27**	**71.56 ± 0.03**	84.01 ± 0.05
NCNC	34.96 ± 3.80	46.93 ± 2.04	>24h	>24h	**91.0 ± 0.24**	**94.72 ± 0.18**	**72.85 ± 0.9**	**86.35 ± 0.51**
NBFNet	OOM	OOM	>24h	>24h	OOM	OOM	OOM	OOM
PEG	30.12 ± 0.63	45.40 ± 0.66	84.21 ± 9.2	95.76 ± 3.48	OOM	OOM	OOM	OOM

Table A.10 Results on ogbl-collab under HeaRT when **excluding the positive train/validation samples** of being negative samples during testing. We highlight the results ranked first, second, and third as green, blue, and orange, respectively.

	MRR	Hits20	Hits50	Hits100
CN	12.60	27.51	38.39	47.4
AA	16.40	32.65	42.61	50.25
RA	28.14	41.16	46.9	51.78
Shortest Path	**46.71**	**46.56**	**46.97**	48.11
Katz	**47.15**	**48.66**	**51.07**	54.28
Node2Vec	12.10 ± 0.20	25.85 ± 0.21	35.49 ± 0.22	46.12 ± 0.34
MF	26.86 ± 1.74	38.44 ± 0.07	43.62 ± 0.08	51.75 ± 0.14
MLP	12.61 ± 0.66	23.05 ± 0.89	35.32 ± 0.74	51.09 ± 0.37
GCN	18.28 ± 0.84	32.90 ± 0.66	43.17 ± 0.36	**54.93 ± 0.14**
GAT	10.97 ± 1.16	29.58 ± 2.42	42.07 ± 1.51	53.45 ± 0.64
SAGE	20.89 ± 1.06	33.83 ± 0.93	43.02 ± 0.63	**54.38 ± 0.27**
GAE	OOM	OOM	OOM	OOM
SEAL	22.53 ± 3.51	36.48 ± 2.55	43.5 ± 1.75	49.25 ± 1.39
BUDDY	**32.42 ± 1.88**	**45.62 ± 0.52**	**50.57 ± 0.18**	**55.63 ± 0.68**
Neo-GNN	21.90 ± 0.65	38.40 ± 0.29	46.93 ± 0.17	53.81 ± 0.19
NCN	17.51 ± 2.50	37.07 ± 2.97	45.89 ± 1.11	52.36 ± 0.33
NCNC	19.02 ± 5.32	35.67 ± 6.78	44.76 ± 4.64	52.41 ± 2.09
NBFNet	OOM	OOM	OOM	OOM
PEG	15.68 ± 1.10	29.74 ± 0.95	38.71 ± 0.17	49.34 ± 0.70

A.9 Additional Investigation on ogbl-collab

As introduced in Section A.7.1, we adopt a different strategy to generate the hard negative samples for ogbl-collab which is a dynamic collaboration graph. In this dataset, nodes represent authors and edges represent a collaboration between two authors. Each edge further includes an attribute that specifies the year of collaboration. Specifically, each edge takes the form of (Author 1, Year, Author 2). The task is to predict collaborations in 2019 (test) based on those until 2017 (training) and 2018 (validation).

Contrary to other datasets, we **do not exclude** the positive training samples when generating the negative samples for validation and test. We note that we also do not exclude positive validation edges when generating the negatives for test. In simpler terms, when creating negative samples for testing, both positive samples from training and validation are considered. This means that negative samples during testing could present in the training and validation positive samples. This approach is reasonable and well-aligned with the real-world scenario in the context of collaboration graphs. Specifically, authors who collaborated in the past might not do so in the future. For

instance, just because the positive sample (Author 2, 2017, Author 3) exists, it does not imply that (Author 2, 2019, Author 3) is also true. However, this is implied to be true if we exclude positive train/validation samples from appearing as negative samples during testing.

We validate this approach by contrasting it with the strategy that **excludes train/validation data when generating hard negatives.** The results are presented in Table A.10. We observed that under this setting, both the Shortest Path and Katz perform considerably well on ogbl-collab. Specifically, the MRR gap between the second-ranked method (Shortest Path) and the third (BUDDY) is 14.29. We found that it is due to the ogbl-collab being a dynamic graph. Of the positive samples in the test set, around 46% also appear as positive samples in the training set. In particular, an edge (Author 1, 2017, Author 2) in the training data may also "appear" in the test data in the form of (Author 1, 2019, Author 2). This is because two authors who collaborated in the past often tend to collaborate again in the future. As such, when evaluating the test sample (Author 1, 2019, Author 2), there exists path of length 1 between the two authors in the graph. Furthermore, this phenomenon is common among positive samples but not observed among negatives. This is because we exclude the positive training samples when generating the negative samples for evaluation. **As a result of this exclusion, the presence of a direct link (i.e., a shortest path of length 1) between two authors suggest a positive sample, while its absence often corresponds to a negative sample.** As such, it provides an easy "shortcut" to distinguish positive and negative samples during testing. This explains why methods like Shortest Path and Katz can achieve good performance on ogbl-collab when excluding positive train/validation samples of being negative samples during testing.

On the contrary, when allowing positive train/validation samples to also be negative samples for HeaRT, the results on ogbl-collab, shown in Tables 2.5 and A.9, indicate that Shortest Path does not maintain its superior performance as observed in Table A.10. Additionally, the overall results under HeaRT are inferior to the ones in Table 2.5. For instance, while all the MRR values in Table 2.5 exceed 10, the highest MRR in Table A.10 is approximately 6. This indicates that excluding those positive samples from being negative samples disproportionately helps the Shortest Path.

A.10 Dataset Licenses

The license for each dataset can be found in Table A.11.

Table A.11 Dataset Licenses.

Datasets	License
Cora	NLM License
Citeseer	NLM License
Pubmed	NLM License
ogbl-collab	MIT License
ogbl-ddi	MIT License
ogbl-ppa	MIT License
ogbl-citation2	MIT License

A.11 Limitation

One potential limitation of HeaRT lies in the generation of customized negative samples for each positive sample. This design may result in an increased number of negative samples compared to the existing setting. Although this provides a more realistic evaluation, it could have an impact on the efficiency of the evaluation process, especially in scenarios where a significant number of positive samples exist. Nonetheless, this limitation does not detract from the potential benefits of HeaRT in providing a more realistic and meaningful link prediction evaluation setting. Furthermore, as each evaluation node pair is independent, it offers scope for parallelization, mitigating any potential efficiency concerns to a large extent. Future work can investigate ways to optimize this process.

A.12 Social Impact

Our method HeaRT harbors significant potential for positive societal impact. By aligning the evaluation setting more closely with real-world scenarios, it enhances the applicability of link prediction research. This not only contributes to the refinement of existing prediction methods but also stimulates the development of more effective link prediction methods. As link prediction has far-reaching implications across numerous domains, from social network analysis to recommendation systems and beyond, improving its performance and accuracy is of paramount societal importance. We also carefully consider the broader impact from various perspectives such as fairness, security, and harm to people. No apparent risk is related to our work.

APPENDIX B

LPFORMER: AN ADAPTIVE GRAPH TRANSFORMER FOR LINK PREDICTION

B.1 Existing Formulations of Pairwise Encodings

In this section we give an overview of existing formulations of pairwise encodings using in DP-MPNNs. The standard formulation of DP-MPNNs is given in Eq. 3.1 where $s(a, b)$ is the pairwise encoding. We briefly describe other existing solutions below:

NCN [121]: NCN only considers the CNs of the target link (a, b) by summing the node representation of each. The pairwise encoding, $s(a, b)$, is written as:

$$s(a, b) = \sum_{u \in \mathcal{N}^{\text{CN}}_{(a,b)}} \mathbf{h}_u, \tag{B.1}$$

where $\mathbf{h}_u$ is the node representation encoded by a MPNN.

NCNC [121]: NCNC extends NCN by further considering the 1-hop neighbors of the node pair that aren't CNs. To account for the difference, they are weighted by the probability of they themselves being CNs of the other node in the pair. This is given for a target link (a, b) as:

$$s(a, b) = \sum_{u \in \mathcal{V}} w(a, b, u)\, \mathbf{h}_u, \tag{B.2}$$

where

$$w(a, b, u) = \left\{ \begin{array}{ll} 1, & \text{when } u \in \mathcal{N}^{\text{CN}}_{(a,b)} \\ \text{NCN}(A, X, b, u) & \text{when } u \in \mathcal{N}(a) \\ \text{NCN}(A, X, a, u) & \text{when } u \in \mathcal{N}(b) \\ 0, & \text{else} \end{array} \right\}. \tag{B.3}$$

This weighting scheme ensures that CNs play a larger role in the pairwise information than non-CNs.

BUDDY [121]: BUDDY considers counting the number of nodes that correspond to different labels given by the double radius node labeling trick [142]. We first define the number of nodes that are a distance d_a and d_b from nodes a and b as $\mathcal{A}_{ab}[d_a, d_b]$. We further define the number of nodes

where $\max(d_u, d_v) > k$ as $\beta_{ab}[d]$. The pairwise encoding concatenates the counts belonging to all combination of $d = 1 \cdots k$. The counts are estimated using subgraph sketching algorithms [35, 16] and are denoted $\hat{\mathcal{A}}$ and $\hat{\mathcal{B}}$. The pairwise encoding for a target link (a, b) is given by the following where $[k] = \{1 \cdots k\}$:

$$s^{\hat{\mathcal{A}}}(a, b) = \Big\|_{d_a, d_b \in [k]} \hat{\mathcal{A}}_{ab}[d_a, d_b], \tag{B.4}$$

$$s^{\hat{\beta}}(a, b) = \Big\|_{d \in [k]} \hat{\beta}_{ab}[d], \tag{B.5}$$

$$s(a, b) = s^{\hat{\mathcal{A}}}(a, b) \Big\| s^{\hat{\beta}}(a, b). \tag{B.6}$$

Neo-GNN [121]: Neo-GNN considers the higher-order neighbor overlap between two nodes. This is done by first learning a structural representation for each node i, x_i^{struct}. This is given by:

$$x_i^{struct} = f_1\left(\sum_{j \in \mathcal{N}(i)} f_2\left(A_{ij}\right)\right). \tag{B.7}$$

To consider the L-hop structural information, the structural representations are diffused over L hops and weighted by a hyperparameter β:

$$Z = \text{MLP}\left(\sum_{l=1}^{L} \beta^{l-1} A^l X^{struct}\right), \tag{B.8}$$

$$\text{where} \quad X = \text{diag}(x^{struct}). \tag{B.9}$$

The pairwise encoding $s(a, b)$ is the dot product of both the final representations,

$$s(a, b) = z_a^T z_b. \tag{B.10}$$

B.2 Special Cases of the General Pairwise Encoding

In this section we demonstrate that multiple popular heuristics and pairwise encodings can be formulated as special cases of the general pairwise encoding given in Eq. (3.2).

Common Neighbors (CNs) [81]: The CNs of a pair of nodes (a, b) is defined the overlapping 1-hop neighbors of both nodes:

$$\mathcal{N}_{(a,b)}^{\text{CN}} = \mathcal{N}(a) \cap \mathcal{N}(b). \tag{B.11}$$

Eq. (3.2) is equal to the CNs when $h(a, b, u) = 1$ and $w(a, b, u)$ is:

$$w(a, b, u) = \left\{ \begin{array}{ll} 1, & \text{when } u \in \mathcal{N}(a) \cap \mathcal{N}(b) \\ 0, & \text{else} \end{array} \right\}. \tag{B.12}$$

Adamic-Adar (AA) [2]: AA is defined as the reciprocal log-degree weighted CN score where d_u is the degree of node u:

$$\text{AA}(a, b) = \sum_{u \in \mathcal{N}^{\text{CN}}_{(a,b)}} \frac{1}{\log(d_u)}. \tag{B.13}$$

Eq. (3.2) can be rewritten as the AA when $h(a, b, u) = 1/\log(d_u)$ and $w(a, b, u)$ is equal to Eq. (B.12).

Resource Allocation (RA) [151]: RA is defined as the reciprocal degree weighted CN score:

$$\text{RA}(a, b) = \sum_{u \in \mathcal{N}^{\text{CN}}_{(a,b)}} \frac{1}{d_u}. \tag{B.14}$$

Eq. (3.2) can be rewritten as the AA when $h(a, b, u) = 1/d_u$ and $w(a, b, u)$ is equal to Eq. (B.12).

Katz Index [49]: The Katz index is a global structural measure. It is defined as weighted summation of the number of paths of different lengths connecting a and b. It is given by the following where the decay weight $\beta \in [0, 1]$,

$$\text{Katz}(a, b) = \sum_{l=1}^{\infty} \beta^l A^l_{a,b}. \tag{B.15}$$

This is equivalent to Eq. (3.2) when:

$$w(a, b, u) = \sum_{l=1}^{\infty} \beta^l e_a^T A^l, \tag{B.16}$$

where $e_i \in \mathbb{B}^{|\mathcal{V}|}$ is a one-hot vector for a node i. We further set,

$$h(a, b, u) = \left\{ \begin{array}{ll} e_b^T, & \text{when } u = b \\ \mathbf{0}, & \text{else} \end{array} \right\}. \tag{B.17}$$

Personalized Pagerank (PPR) Score [15]: The personalized pagerank score is the pagerank score localized to a root node u. The localization is via a teleportation probability α that transports the

random walk back to the root node. We show that Eq. (3.2) can be rewritten as the PPR score when setting $h(a, b, u)$ equal to (B.17) and, following [24], setting $w(a, b, u)$ to:

$$w(a, b, u) = \alpha \sum_{l=0}^{\infty} (1 - \alpha)^l e_a^T (D^{-1} A)^l. \tag{B.18}$$

Feature Similarity: The feature similarity of the pair of nodes (a, b) is expressed by $\text{dis}(\mathbf{x}_a, \mathbf{x}_b)$ where $\mathbf{x}_a$ are the node features of node a and $\text{dis}(\cdot)$ is a distance function (e.g., euclidean distance). This can be rewritten as Eq. (3.2) by substituting:

$$w(a, b, u) = \text{dis}(\mathbf{x}_a, \mathbf{x}_u), \tag{B.19}$$

and $h(a, b, u) = e_b^T$ where $e_i \in \mathbb{B}^{|\mathcal{V}|}$ is a one-hot vector for a node i.

NCN [121]: The pairwise encoding used in NCN is defined as the summation of the representations for the CNs of a link. Eq. (3.2) can be rewritten as NCN when $w(a, b, u)$ is equal to Eq. (B.12). $h(a, b, u)$ is equal to the node representation u encoded by a MPNN, i.e., $h(a, b, u) = \mathbf{h}_u$ where $H = \text{MPNN}(A, X)$.

NCNC [121]: NCNC extends NCNC by further weighting the 1-hop (non-CN) by their probability of linking to the other nodes. Given Eq. (3.2), the weight $w(a, b, u)$ is equal to following where 1-hop neighbors are weighted by their probability of linking with the other node:

$$w(a, b, u) = \left\{ \begin{array}{ll} 1, & \text{when } u \in \mathcal{N}_{(a,b)}^{\text{CN}} \\ \text{NCN}(A, X, b, u) & \text{when } u \in \mathcal{N}(a) \\ \text{NCN}(A, X, a, u) & \text{when } u \in \mathcal{N}(b) \\ 0, & \text{else} \end{array} \right\}. \tag{B.20}$$

$\text{NCN}(A, X, a, u)$ is the probability of a and u being linked using the NCN model. We further define $h(a, b, u) = \mathbf{h}_u$.

Neo-GNN [136]: The pairwise encoding used in Neo-GNN considers the higher-order neighborhood overlap between two nodes. The formulation is given in Section B.2. When $l = 1$, it can be

expressed using Eq. (3.2) by setting:

$$h(a,b,u) = f_1\left(\sum_{v\in\mathcal{N}(u)} f_2\left(A_{uv}\right)\right)^2, \tag{B.21}$$

and

$$w(a,b,u) = \left\{ \begin{array}{ll} 1, & \text{when } u \in \mathcal{N}^{\text{CN}}_{(a,b)} \\ 0, & \text{else} \end{array} \right\}. \tag{B.22}$$

B.3 Proof of Proposition 1

Proposition 1. *Consider a target link (a,b) and a node $u \in \mathcal{V} \setminus \{a,b\}$. The PPR [15] score of a root node i and target node j with teleportation probability α is denoted by $ppr(i,j)$. Let $r_a^k(u)$ be the probability of a walk of length k beginning at node a and terminating at u. We define $r_{a,b}^k(u) := r_a^k(u) + r_b^k(u)$. We also define a weight $\gamma^k := \alpha(1-\alpha)^k$ for all walks of length k. The PPR scores, $ppr(a,u)$ and $ppr(b,u)$, along with the random walk probabilities of disparate lengths, are interconnected through the following relationship.*

$$\Gamma(a,b,u) = ppr(a,u) + ppr(b,u) = \sum_{k=0}^{\infty} \gamma^k r_{a,b}^k(u). \tag{3.6}$$

Proof. Per [24], the PPR vector for a root node s, pr_s, is equivalent to:

$$\text{pr}_s = \alpha \sum_{k=0}^{\infty} (1-\alpha)^k W^k x_s, \tag{B.23}$$

where W is a the random walk matrix and x_s is a preference vector that is a one-hot vector for element s. We note that $\text{pr}_s(t)$ represents the landing probability of node t given the root node s. As such, by definition, $\text{pr}_s(t) = \text{ppr}(s,t)$. Furthermore, it is clear that $r_s^k = W^k x_s \in \mathbb{R}^{\mathcal{V}}$ represents the probability of a walk of length k beginning at node s and stop all other nodes, individually. Also, the probabilities of all walks of length k are weighted by $\gamma^k = \alpha(1-\alpha)^k$. $\Gamma(a,b,u)$ can be obtained by first taking the sum of the PPR vectors for nodes a and b,

$$\begin{aligned} \text{pr}_a + \text{pr}_b &= \alpha \sum_{k=0}^{\infty} (1-\alpha)^k W^k x_a + \alpha \sum_{k=0}^{\infty} (1-\alpha)^k W^k x_b, \\ \text{pr}_{a,b} &= \alpha \sum_{k=0}^{\infty} (1-\alpha)^k W^k \left(x_a + x_b\right), \end{aligned} \tag{B.24}$$

where $\text{pr}_{a,b} = \text{pr}_a + \text{pr}_b$. From this, we can express $\Gamma(a, b, u)$ as:

$$\begin{aligned}\Gamma(a, b, u) &= \text{ppr}(a, u) + \text{ppr}(b, u), \\ &= \text{pr}_{a,b}(u), \\ &= \text{pr}_a(u) + \text{pr}_b(u),\end{aligned} \tag{B.25}$$

which as shown in Eq. (B.24) is equivalent to the probability of a walk that originates from either node a or b and terminates at node u. This completes the proof. □

B.4 Attention Formulation

For a target link (a, b), LPFormer attends to the nodes in the set $\bar{\mathcal{V}}(a, b)$. The attention mechanism used in LPFormer is defined in Section 5.3 as follows where $w(a, b, u)$ is the attention weight of u to the target link and $\bar{\mathcal{V}}(a, b) = \mathcal{V} \setminus \{a, b\}$:

$$\begin{aligned}\tilde{w}(a, b, u) &= \phi\left(\mathbf{h}_a, \mathbf{h}_b, \mathbf{h}_u,\ \mathbf{rpe}_{(a,b,u)}\right), \\ w(a, b, u) &= \frac{\exp(\tilde{w}(a, b, u))}{\sum_{v \in \bar{\mathcal{V}}(a,b)} \exp(\tilde{w}(a, b, u))}.\end{aligned} \tag{B.26}$$

The function $\phi(\cdot)$ is modeled via the attention mechanism defined in GATv2 [17]. We define $a \in \mathbb{R}^{2d'}$ and $W \in \mathbb{R}^{d \times d'}$. The raw attention weights are then given by:

$$\tilde{w}(a, b, u) = \mathbf{a}^T \text{ LeakyReLU}\left[W\,\mathbf{h}_a \,\middle\|\, W\,\mathbf{h}_b \,\middle\|\, W\,\mathbf{h}_u \,\middle\|\, \mathbf{rpe}_{(a,b,u)}\right]. \tag{B.27}$$

The final attention weights, $w(a, b, u)$, are given by passing $\tilde{w}(a, b, u)$ through a softmax activation layer.

B.5 Additional Experimental Details

B.5.1 Planetoid splits

We note that for each of Cora, Citeseer, Pubmed we use a fixed split. This follows the recent work of [61]. [61] observe that for Cora, Citeseer, Pubmed there exists no unified data split between studies. They find that while recent work [20, 121] use 10 random splits, prior work [154, 115] use a fixed split and train over 10 random seeds. Furthermore, there exists discrepancies in the preprocessing between those works that use the random splits. [20] only use the largest connected

component of each dataset while [121] use the whole dataset. This makes any comparison of the published results difficult. Due to these discrepancies, we use the performance on the fixed split given by [61], as *it's the only split where all methods are evaluated and compared under the same setting*.

B.5.2 Omission of ogbl-ddi under the Existing Evaluation

We further omit the results of ogbl-ddi in Table 4.1. This is due to the observation made by [61] that there exists a poor relationship between the validation and test performance. This extends to recent pairwise MPNNs, including NCN [121], Neo-GNN [136], and BUDDY [20]. This makes tuning on the validation set difficult, as it doesn't guarantee good test performance. Due to this, they observe that when tuning on a fixed set of hyperparameter ranges, they are unable to achieve comparable results to the reported performance. Often they observe that the performance is actually much lower. Due to these concerns we believe ogbl-ddi is not suitable for the task of transductive link prediction and don't report the performance. For more details and discussion, please see Appendix D in [61]. However, they show that this problem does not afflict ogbl-ddi under the newly proposed HeaRT [61] evaluation setting. As such, we further include the results for our method under HeaRT in Table 3.5.

B.5.3 Computation of the PPR Matrix

We compute the PPR matrix via the efficient approximation algorithm introduced by [5]. The estimation is controlled by a tolerance parameter ϵ. The parameter ϵ controls both the speed of computation and the sparsity of the solution (i.e., a higher value of ϵ will produce a sparser PPR matrix). We use: $\epsilon = 1e^{-7}$ for Cora and Citeseer, $\epsilon = 5e^{-5}$ for ogbl-collab and ogbl-ppa, $\epsilon = 1e-5$ for Pubmed, and $\epsilon = 5e^{-3}$ for ogbl-Citation2. The value of ϵ is chosen as a trade-off between accuracy and sparsity to allow for ease of storage in GPU memory.

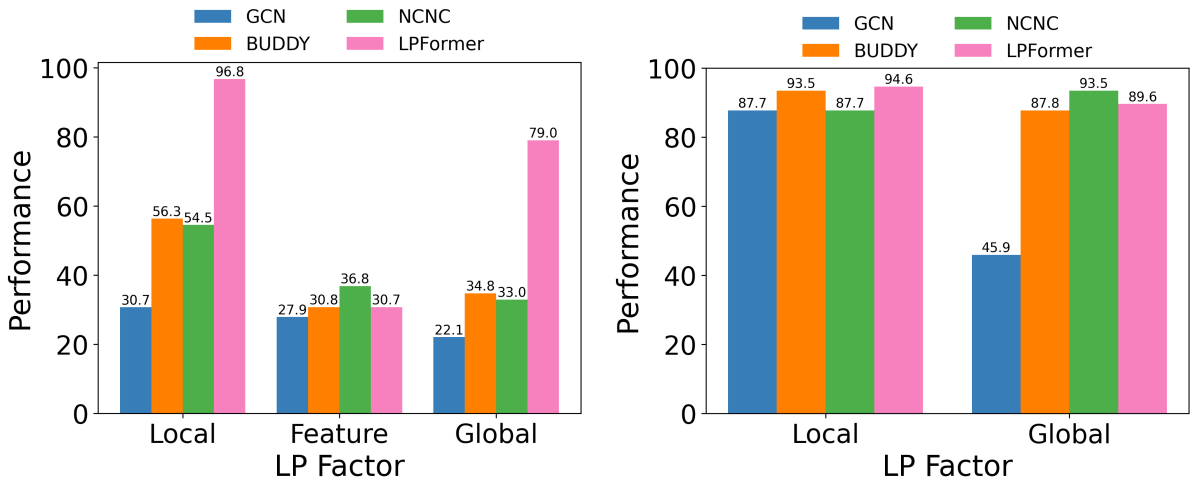
B.5.4 Splitting Target Links by LP Factor

In Section 3.4.3 we demonstrate the performance on samples that correspond to a single LP factor. In this section we further detail the algorithm used to determine the set of samples corresponding to each factor. We consider the three main factors: local structural information,

global structural information, and feature proximity. We measure each using a single representative heuristic: CNs [81] for local information, PPR [15] for global information, and cosine feature similarity for feature proximity. For each sample, we check if the score is only high in **one** heuristic. In this way, it tells us that there is a dominant factor present in the pairwise information.

This determination is done by comparing the the heuristic scores of each target link against a threshold value. For a LP factor i and target link (a, b), we denote the heuristic score as $s^i(a, b)$. The threshold value for factor i is represented by $\hat{s}^i$ and is chosen such that it corresponds to a higher score. We desire $\hat{s}^i$ to be a higher score such that any score $\geq$ than it indicates that a plethora of pairwise information exists corresponding to factor i. This is done by setting the threshold equal to the p-th percentile value for that heuristic among all target links. For example, for CNs, the 80th percentile score on one dataset may be 9. The value of p is chosen to be high (e.g., 80%) due to the aforementioned reasoning. Given these inputs, for each target link we compare the score for factor i against the threshold value of that factor. Continuing our example, if (a, b) only has 2 CNs, it is below the previously defined threshold. We only consider a sample as "belonging" to a single factor when it is $s^i(a, b) \geq \hat{s}^i$ is true **for one only one factor** i. So if the heuristic score for (a, b) is below the p-th percentile threshold for CNs and PPR but above for feature similarity, then feature proximity will be considered the dominant LP factor. However, if it's above the threshold for both local and structural information, it will not be assigned to any group. This is done as we want to isolate links that only highly express one LP factor. This allows us to better understand how certain methods can model that specific factor. The detailed algorithm is given in Algorithm B.1.

We note that each target link may not belong to a category. This can be due to there being no or many dominant LP factor. We further set the percentile equal to 90% on all datasets except for ogbl-collab for which we use 80%. These values were chosen as we wanted the percentile to be suitably high such that we are confident that the corresponding factor is relevant to the target link. Furthermore, we use a lower value for ogbl-collab as we found it produced a more even distribution of links by factor.



(a) Pubmed (b) ogbl-ppa

Figure B.1 Performance for target links when there is only one LP factor strongly expressed. Results are on (a) Pubmed, (b) ogbl-ppa. We note that due the quality of features used, we omit the feature proximity factor for ogbl-ppa from our analysis.

B.5.5 Additional Results for the LP Factor Experiments

In Section 3.4.3 we observed the performance of various methods on target links where only a single LP factor is expressed. This is done through the use of heuristic scores. We further demonstrate the results on the Pubmed and ogbl-ppa datasets. Of note is that for ogbl-ppa the initial node features are one-hot vectors that signify the species that the protein belongs to. We observe that due to the sparseness of these features, feature proximity measures are unable to properly predict any target links on their own. As such, the factor corresponding to feature proximity is not expressed. We therefore exclude that factor for this analysis on ogbl-ppa.

The results for both Pubmed and ogbl-ppa datasets are given in Figure B.1. As shown earlier in Figure 3.3, LPFormer can most consistently perform well across each factor. This suggests that LPFormer is best able to both model a variety of factors and adapt accordingly for each target link.

B.5.6 Performance on Heterophilic Datasets

In this section we evaluate LPFormer on multiple heterophilic datasets. Heterophily refers to the tendency of dissimilar nodes to be connected. This is as opposed to homophily, in which nodes with similar attributed are more likely to be connected. Since most graphs used for benchmark

datasets tend to contain homophilic patterns, heterophilic graphs present an interesting challenge regarding the effectiveness of graph-based methods. For a more detailed discussion on heterophilic graphs, please see [71].

We test on two prominent heterophilic datasets, Squirrel and Chameleon [94]. The statistics for each are in Table B.1. We limit our comparison to those LP methods that tend achieve the best results, including GCN, BUDDY, and NCNC. In Table B.2, we report the MRR over five random seeds. Note that we test under the **original evaluation setting** and not HeaRT. We observe that LPFormer can achieve a large increase over other methods, with a 14% and 9% increase in performance on Squirrel and Chameleon, respectively. These results indicate the superior ability of LPFormer to accurately model LP on heterophilic graphs, as compared to other methods.

Table B.1 Heterophilic Dataset Statistics.

	Squirrel	Chameleon
#Nodes	5201	2277
#Edges	198,353	31,371
Split Ratio	85/5/10	85/5/10

Table B.2 Results on Heterophilic Datasets.

Method	**Squirrel**	**Chameleon**
GCN	22.77 ± 4.54	20.74 ± 8.08
BUDDY	9.69 ± 0.99	6.30 ± 2.40
NCNC	32.37 ± 5.46	26.24 ± 3.37
LPFormer	**36.77 ± 2.77**	**28.61 ± 6.68**
% Improvement	14%	9%

B.5.7 More Efficiently Incorporating the PPR Scores

In Figure 3.4 we compare the training time between LPFormer and NCNC. We observe that on the denser datasets, ogbl-ppa and ogbl-ddi, LPFormer is considerably more efficient. Furthermore, on ogbl-collab, both methods have a fast runtime. However, we find that LPFormer struggles on ogbl-citation2 in comparison to NCNC. We observe that this is due to the need of the PPR matrix, which while sparse, requires a large amount of memory and processing time. In the future, we plan

Algorithm B.1 Determining Samples by LP Factor

Require:
$\text{CN}(\cdot)$ = Maps (i, j) to # of CNs of the pair
$\text{PPR}(\cdot)$ = Maps (i, j) to PPR score of the pair
$\text{FS}(\cdot)$ = Maps (i, j) to feature cosine similarity of the pair
p = Percentile used to determine whether a factor is present
$\mathcal{E}^{\text{test}}$ = Positive test links

// Compute the score corresponding to the p-th percentile for each heuristic
$\hat{s}^{\text{CN}} = \text{Percentile}(p, \{CN(i, j) \mid (i, j) \in \mathcal{E}^{\text{test}}\})$
$\hat{s}^{\text{FS}} = \text{Percentile}(p, \{FS(i, j) \mid (i, j) \in \mathcal{E}^{\text{test}}\})$
$\hat{s}^{\text{PPR}} = \text{Percentile}(p, \{PPR(i, j) \mid (i, j) \in \mathcal{E}^{\text{test}}\})$

Create empty lists L^{CN}, L^{PPR}, and L^{FS}
for $(i, j) \in \mathcal{E}^{\text{test}}$ **do**
    link-cn = $\text{CN}(i, j)$
    link-fs = $\text{FS}(i, j)$
    link-ppr = $\text{PPR}(i, j)$

    // Assign sample to corresponding list based on scores
    if link-cn $\geq \hat{s}^{\text{CN}}$ **and** link-fs $< \hat{s}^{\text{FS}}$ **and** link-ppr $< \hat{s}^{\text{PPR}}$ **then**
        Append(L^{CN}, (i, j))
    else if link-cn $< \hat{s}^{\text{CN}}$ **and** link-fs $\geq \hat{s}^{\text{FS}}$ **and** link-ppr $< \hat{s}^{\text{PPR}}$ **then**
        Append(L^{FS}, (i, j))
    else if link-cn $< \hat{s}^{\text{CN}}$ **and** link-fs $< \hat{s}^{\text{FS}}$ **and** link-ppr $\geq \hat{s}^{\text{PPR}}$ **then**
        Append(L^{PPR}, (i, j))
    end if
end for
return L^{CN}, L^{PPR}, L^{FS}

to fix this problem by performing a simple and efficient pre-processing step. Specifically, before training, we can iterate over all target links and extract the relevant PPR scores. This would obviate the need to store the PPR matrix and determine the nodes for each link. Furthermore, this only needs to be done once before tuning the model. This would greatly reduce the storage and time needed to train LPFormer on all datasets and is an avenue we plan to explore in the future.

APPENDIX C

TOWARD DEGREE BIAS IN EMBEDDING-BASED KNOWLEDGE GRAPH COMPLETION

C.1 Dataset Statistics

The statistics for each dataset can be found in Table C.1.

Table C.1 Dataset Statistics.

Statistic	FB15K-237	NELL-995	CoDEx-M
#Entities	14,541	74,536	17,050
#Relations	237	200	51
#Train	272,115	149,678	185,584
#Validation	17,535	543	10,310
#Test	20,466	2,818	10,311

C.2 Infrastructure

All experiments were run on a single 32G Tesla V100 GPU and implemented using PyTorch [86].

C.3 Parameter Settings

Each model is trained for 400 epochs on FB15K-237, 250 epochs on CoDEx-M, and 300 epochs on NELL-995. The embedding dimension is set to 200 for both methods except for the dimension of the relation embeddings in TuckER which is is tuned from $\{50, 100, 200\}$. The batch size is set to 128 and the number of negative samples per positive sample is 100. The learning rate is tuned from $\{1e^{-5}, 5e^{-5}, 1e^{-4}, 5e^{-4}, 1e^{-3}\}$, the decay from $\{0.99, 0.995, 1\}$, the label smoothing from $\{0, 0.1\}$, and the dropout from {0, 0.1, 0.2, 0.3, 0.4, 0.5 }. For KG-Mixup we further tune the degree threshold from $\{3, 5, 10\}$, the number of samples generated from $\{5, 10\}$, and the loss weight for the synthetic samples from {1e-2, 1e-1, 1 }. Lastly, we tune the stochastic weight averaging (SWA) initial learning rate from { 1e-5, 5e-4 }. The best hyperparameter values for ConvE and TuckER using KG-Mixup are shown in Figures C.2 and C.3, respectively.

C.4 Preliminary Study Results using ConvE

In Section 4.3 we conduct a preliminary study using TuckER [7] on the FB15K-237 dataset [111]. In this section we further include the corresponding results when using the ConvE [29] embedding models. The plots can be found in Figure C.1. We note that they show a similar pattern to those displayed by TuckER in Figure 4.2.

Table C.2 Hyperparameter values for ConvE on each dataset.

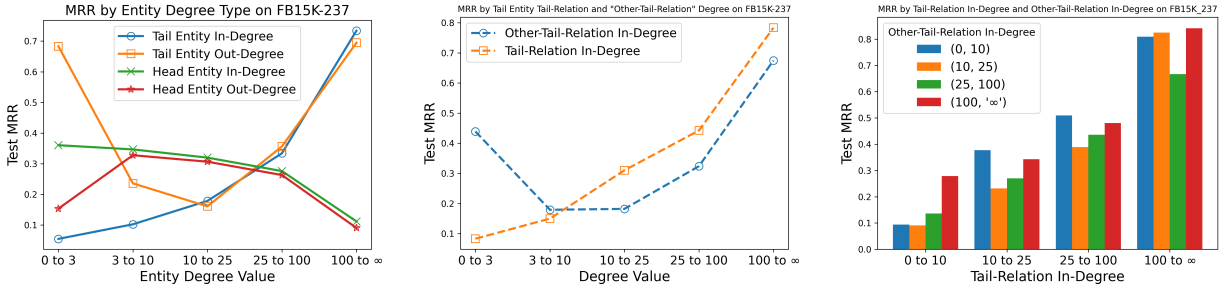
Hyperparameter	**FB15K-237**	**NELL-995**	**CoDEx-M**
Learning Rate	$1e^{-4}$	$5e^{-5}$	$1e^{-5}$
LR Decay	None	None	None
Label Smoothing	0	0.1	0.1
Dropout #1	0.2	0.4	0.1
Dropout #2	0.5	0.3	0.2
Dropout #3	0.2	0.1	0.3
Degree Threshold	5	5	5
# Generated	5	5	5
Synth Loss Weight	1	1	1
SWA LR	$5e^{-4}$	$1e^{-5}$	$1e^{-5}$

Table C.3 Hyperparameter values for TuckER on each dataset.

Hyperparameter	**FB15K-237**	**NELL-995**	**CoDEx-M**
Learning Rate	$5e^{-5}$	$5e^{-5}$	$1e^{-5}$
LR Decay	0.99	None	0.995
Label Smoothing	0	0	0
Dropout #1	0.3	0.3	0.3
Dropout #2	0.4	0.3	0.5
Dropout #3	0.5	0.2	0.5
Rel Dim	200	100	100
Degree Threshold	5	25	5
# Generated	5	5	5
Synth Loss Weight	1	$1e^{-2}$	1
SWA LR	$5e^{-4}$	$1e^{-5}$	$5e^{-4}$

C.5 Expected Calibration Error

Expected calibration error [40] is a measure of model calibration that utilizes the model accuracy and prediction confidence. Following [40], we first split our data into M bins and define the accuracy



(a) In and Out Degree Analysis (b) Tail In-Degree Analysis (c) Controlling for Degrees

Figure C.1 MRR when predicting the tail for ConvE on FB15K-237 when varying the (a) in-degree and out-degree of the head and tail entity, (b) tail-relation and other-relation in-degree, and (c) other-relation in-degree for smaller sub-ranges of the tail-relation degree.

(acc) and confidence (conf) on one bin B_m as:

$$\text{acc}(B_m) = \frac{1}{|B_m|} \sum_{i=1}^{|B_m|} \mathbf{1}(\hat{y}_i = y_i), \tag{C.1}$$

$$\text{conf}(B_m) = \frac{1}{|B_m|} \sum_{i=1}^{|B_m|} \hat{p}_i, \tag{C.2}$$

where y_i is the true label for sample i, $\hat{y}_i$ is the predicted label, and $\hat{p}_i$ the prediction probability for sample i. A well-calibrated model should have identical accuracy and confidence for each bin. ECE is thus defined by [40] as:

$$\text{ECE} = \sum_{m=1}^{M} \frac{|B_m|}{n} |\text{acc}(B_m) - \text{conf}(B_m)|, \tag{C.3}$$

where n is the total number of samples across all bins. As such, a lower ECE indicates a better calibrated model. For KGC we split the samples into bin by the tail-relation degree. Furthermore to calculate the accuracy score (acc) we utilize His@10 to denote a correct prediction. For the confidence score (conf) we denote $\sigma(f(h,r,t))$ as the prediction probability where $f(h,r,t)$ is a KG embedding score function and σ is the sigmoid function. When calculating the ECE over all samples Eq. (C.3) is unchanged. When calculating ECE for just one bin of samples (e.g. low degree triples) it is defined as:

$$\text{ECE}_m = |\text{acc}(B_m) - \text{conf}(B_m)|, \tag{C.4}$$

where ECE_m is the expected calibration error for just bin m.

C.6 Proof of Theorem 1

In this section we prove Theorem 1. Following recent work [19, 138] we examine the regularizing effect of the mixup parameter λ. This is achieved by approximating the loss l_θ on a single mixed sample $\tilde{e}$ using the first-order quadratic Taylor approximation at the point $\tau = 1 - \lambda$ near 0. Assuming l_θ is differentiable, we can approximate l_θ, with some abuse of notation as:

$$l_\theta(\tau) = l_\theta(0) + l'_\theta(0)\tau. \tag{C.5}$$

We consider the case where l_θ is the binary cross-entropy loss. Since the label $y_{ij} = 1$ is always true, it can be written as,

$$l_\theta(\tilde{e}) = log\ \sigma\left(f(\tilde{e})\right), \tag{C.6}$$

where σ is the sigmoid function and $\tilde{e} = \left(\tilde{h}, \tilde{r}, t\right)$ is the mixed sample. Since $\tau = 1 - \lambda$, we can rewrite the mixed sample as:

$$\tilde{e} = ((1-\tau)h_i + \tau h_j, (1-\tau)r_i + \tau r_j, t). \tag{C.7}$$

As such, setting $\tau = 0$ doesn't mix the two samples resulting in $\tilde{e} = (h_i, r_i, t)$. The term $l_\theta(0)$ is therefore equivalent to the standard loss $\mathcal{L}(\theta)$ over the samples S. We can now compute the first derivative in Eq. (C.5). We evaluate l'_θ via the chain rule,

$$l'_\theta = \frac{\partial\ log\ \sigma\left(f(\tilde{e})\right)}{\partial \sigma\left(f(\tilde{e})\right)} \cdot \frac{\partial \sigma\left(f(\tilde{e})\right)}{\partial f(\tilde{e})} \cdot \frac{\partial f(\tilde{e})}{\partial \tau}, \tag{C.8}$$

where $\frac{\partial f(\tilde{e})}{\partial \tau}$ is evaluated via the multivariable chain rule to:

$$\frac{\partial f(\tilde{e})}{\partial \tau} = \left(\frac{\partial f(\tilde{e})}{\partial x_{\tilde{h}}}\frac{\partial x_{\tilde{h}}}{\partial \tau} + \frac{\partial f(\tilde{e})}{\partial x_{\tilde{r}}}\frac{\partial x_{\tilde{r}}}{\partial \tau} + \frac{\partial f(\tilde{e})}{\partial x_t}\frac{\partial x_t}{\partial \tau}\right). \tag{C.9}$$

Evaluating l'_θ:

$$l'_\theta = \frac{1}{\sigma\left(f\left(\tilde{e}\right)\right)} \cdot \sigma\left(f\left(\tilde{e}\right)\right)\left(1 - \sigma\left(f\left(\tilde{e}\right)\right)\right) \cdot \frac{\partial f(\tilde{e})}{\partial \tau}, \tag{C.10}$$

$$= \left(1 - \sigma\left(f\left(\tilde{e}\right)\right)\right) \cdot \frac{\partial f(\tilde{e})}{\partial \tau}, \tag{C.11}$$

$$= \left(1 - \sigma\left(f\left(\tilde{e}\right)\right)\right)\left[\frac{\partial f\left(\tilde{e}\right)}{\partial x_{\tilde{h}}}(x_{h_j} - x_{h_i}) + \frac{\partial f\left(\tilde{e}\right)}{\partial x_{\tilde{r}}}(x_{r_j} - x_{r_i})\right], \tag{C.12}$$

where the term related to x_t in Eq. (C.9) cancels out since $\partial x_t / \partial \tau = 0$. Since we are evaluating the expression near $\tau = 0$, we only consider the original sample in the score function, reducing the above to the following where $\Delta h = (x_{h_j} - x_{h_i})$, and $\Delta r = (x_{r_j} - x_{r_i})$,

$$l'_\theta(0) = (1 - \sigma\left(f\left(e_i\right)\right)) \left[\frac{\partial f\left(e_i\right)}{\partial x_{\tilde{h}}} \Delta h + \frac{\partial f\left(e_i\right)}{\partial x_{\tilde{r}}} \Delta r \right]. \tag{C.13}$$

Plugging in $l_\theta(0)$ and $l'_\theta(0)$ into Eq. (C.5) and rearranging the terms we arrive at:

$$\mathcal{L}_{\text{Mix}}(\theta) = \mathcal{L}(\theta) + \mathcal{R}_1(\theta) + \mathcal{R}_2(\theta), \tag{C.14}$$

where $\mathcal{R}_1(\theta)$ and $\mathcal{R}_2(\theta)$ are defined over all samples S as:

$$\mathcal{R}_1(\theta) = \frac{\tau}{|S|} \sum_{i=1}^{|S|} \sum_{j=1}^{k} \left(1 - \sigma\left(f(e_i)\right)\right) \frac{\partial f(e_i)^T}{\partial x_{\tilde{h}}} \Delta h, \tag{C.15}$$

$$\mathcal{R}_2(\theta) = \frac{\tau}{|S|} \sum_{i=1}^{|S|} \sum_{j=1}^{k} \left(1 - \sigma\left(f(e_i)\right)\right) \frac{\partial f(e_i)^T}{\partial x_{\tilde{r}}} \Delta r, \tag{C.16}$$

with $\tau = \mathbb{E}_{\lambda \sim \mathcal{D}_\lambda}(1 - \lambda)$ where $\mathcal{D}_\lambda = \text{Beta}(\alpha, \alpha)$.

APPENDIX D

DISTANCE-BASED PROPAGATION FOR EFFICIENT KNOWLEDGE GRAPH REASONING

D.1 Proof Details of Theorem 2

We prove Theorem 2 via induction on the path length l. We denote all nodes a distance l from the source node s as V_s^l. The path length offset is represented by δ. Lastly, for convenience we split the constraints in Eq. (5.6) into two: a *node constraint* and an *edge constraint*. We formulate it as the following where $\text{Node}_\delta(s, o, t)$ represents the node constraint and $\text{EdgeC}_\delta(s, o, u)$ the edge constraint:

$$\text{Node}_\delta(s, o, t) = t - \delta \leq \text{dist}(s, o) \leq t, \tag{D.1}$$

$$\text{EdgeC}_\delta(s, o, u) = \text{dist}(s, u) < \text{dist}(s, o) + \delta \tag{D.2}$$

Base Case (l=1): We want to show for all $l = 1$ hop neighbors of s, $o \in V_s^1$, their final representation $x_q^F(s, o)$ aggregates all path representations in the range $[0, 1{+}\delta]$. To be true, the embedding $x_q^F(s, o)$ must satisfy two conditions:

- *Condition 1*: The final embedding $x_q^F(s, o)$, contains all paths representations of length less than or equal to $1 + \delta$ between s and o.
- *Condition 2*: The final embedding $x_q^F(s, o)$ contains no other path information.

Condition 1: For it to be true, a node $o \in V_s^1$ must aggregate all edges of the form (u, r, o) where u belongs to the set:

$$U_{s,o}^{(0,\delta)} = \{u \mid (u, r, o) \in \mathcal{E}_o,\ u \in \{V_s^0, V_s^1, \cdots, V_s^\delta\}\}, \tag{D.3}$$

where $\mathcal{E}_o$ represents all edges where o is the target node. It's intuitive that all paths starting at s of length $\in [0, \delta + 1]$ must pass through the nodes in the set $U_{s,o}^{(0,\delta)}$ in order to reach o. We prove in Theorem 3 that o will aggregate all nodes in the set $U_{s,o}^{(0,\delta)}$.

Condition 2: We want to demonstrate that the representation of node o aggregates no other path information such that $x_q^{(\delta+1)}(s, o) = x_q^F(s, o)$. This is true as per the node constraint (Eq. (D.1)) the representation of a node o stops updating after iteration $k = 1 + \delta$.

Inductive Step: We assume that for all m-hop neighbors of s, $o \in V_s^m$, their final representation $x_q^F(s,o)$ aggregates all path representations of length between $[m, m+\delta]$. This is achieved by a node o aggregating all edges (u,r,o) where u belongs to the set:

$$U_{s,o}^{(m-1,m-1+\delta)} = \{u \mid (u,r,o) \in \mathcal{E}_o,\ u \in \{V_s^{m-1}, \cdots, V_s^{m-1+\delta}\}\}, \tag{D.4}$$

as all such paths must pass through these nodes. We note that this implies that:

- The set of nodes $U_{s,o}^{(m-1,m-1+\delta)}$ must themselves only contain all path representations of lengths $[m-1, m-1+\delta]$ when aggregated by $o \in V_s^m$.
- The set of nodes $U_{s,o}^{(m-1,m-1+\delta)}$ must obtain such path information by iteration $k = m-1+\delta$. This must be true as per the node constraint o will last update at iteration $k = m+\delta$.

We now want to show for all $(m+1)$ hop neighbors of s, $o \in V_s^{m+1}$, their final representation $x_q^F(s,o)$ aggregates all path representations of of length between $[m+1, m+1+\delta]$. This requires showing that $x_q^F(s,o)$ (1) contains all paths representations between $[m+1, m+1+\delta]$ between s and o and (2) it contains no other path information.

<u>*Condition 1*</u>: For $o \in V_s^{m+1}$ to aggregate all paths of length between $m+1$ and $m+1+\delta$, their representation must aggregate all edges (u,r,o) where u belongs to the set:

$$U_{s,o}^{(m,m+\delta)} = \{u \mid (u,r,o) \in \mathcal{E}_o,\ u \in \{V_s^{m}, \cdots, V_s^{m+\delta}\}\}. \tag{D.5}$$

Such edges are aggregated by $o \in V_s^{m+1}$ via the edge constraint. Furthermore,

- From the inductive step we know that nodes $U_{s,o}^{(m-1,m-1+\delta)} = U_{s,v}^{(m,m+\delta)} \setminus V_s^{m+\delta}$ have already aggregated all path representations of lengths $[m-1, m-1+\delta]$ by iteration $k = m+\delta$.
- From both constraints we know that $\forall u \in V_s^{m+\delta}$ will only contain all path representations of length $m+\delta$ (i.e. shortest path) by iteration $k = m+\delta$.

As such, after aggregating the nodes in the set $U_{s,o}^{(m,m+\delta)}$ the representation $x_q^{(m+\delta)}(s,u)$ will contain all paths representations between m and $m+\delta$. Per the node constraint, $\forall o \in V_s^{m+1}$ last update at iteration $k = m+1+\delta$. Therefore by aggregating $U_{s,o}^{(m,m+\delta)}$ at iteration $k = m+1+\delta$, the representation $x_q^{(m+1+\delta)}(s,o)$ will contain all path representations between length $m+1$ and $m+1+\delta$.

Condition 2: Lastly, we want to show that $\forall o \in V_s^{m+1}$ the final representation $x_q^F(s,o)$ will only contain path representations of length $m+1$ to $m+1+\delta$. This is true as per the node constraint the representation of a node $o \in V_s^{m+1}$ last updates at iteration $k = m+1+\delta$. Therefore $x_q^{(m+1+\delta)}(s,o) = x_q^F(s,o)$. As such, the final representation only aggregates paths of length between $m+1$ and $m+1+\delta$.

Theorem 3. *We are given a source node s, query q, and target node o which is a 1-hop neighbor of s. The final representation of a 1-hop neighbor o,* $\mathbf{x}_q^F(s,o)$*, will* ***at minimum*** *aggregate all path representations whose path length is between* 1 *and* $1+\delta$*. It therefore* ***at least*** *contains the path information,*

$$\eta = \bigoplus_{l=1}^{1+\delta} \bigoplus_{p \in P_{s,o}^l} \bigotimes_{i=1}^{|p|} w(e_i). \tag{D.6}$$

This is equivalent to stating that o will aggregate all nodes in the following set by iteration $k = 1+\delta$*,*

$$U_{s,o}^{(0,\delta)} = \{u \mid (u,r,o) \in \mathcal{E}_o,\ u \in \{V_s^0, V_s^1, \cdots, V_s^\delta\}\}. \tag{D.7}$$

We prove this Theorem via induction on the layer iteration k in our algorithm D.1 (denoted their as l).

Base Case (k=1): We want to first show that after one iteration, the representation of a 1-hop neighbor $x_q^1(s,o)$ aggregates all paths of length 1 from the source. This is achieved by $x_q^1(s,o)$ aggregating all edges connecting o to s, i.e. (s,r,o). Such edges are aggregated by o as both the edge and node constraints are satisfied:

$$\text{EdgeC}_\delta(s,o,s) = 0 < 1+\delta, \tag{D.8}$$

$$\text{NodeC}_\delta(s,o,1) = 1-\delta \leq 1 \leq 1. \tag{D.9}$$

Inductive Step: We assume that at some iteration $k = n$, s.t. $n < 1+\delta$, the representation $x_q^n(s,o)$ for $o \in V_s^1$ aggregates all path representations up to a length n from the source. This is achieved by aggregating all edges that contain nodes in the set:

$$U_{s,o}^{(0,n-1)} = \{u \mid (u,r,o) \in \mathcal{E}_o,\ u \in \{V_s^0, V_s^1, \cdots, V_s^{n-1}\}\}. \tag{D.10}$$

Since we assume that $x_q^n(s,o)$ contains all path representations up to length n, then it follows that $\forall u \in U_{s,o}^{(0,n-1)}$ their corresponding representation $x_q^n(s,o)$ must also contain all paths up to length $n-1$. As such, by node o aggregating $U_{s,o}^{(0,n-1)}$ it extend the length of each path by 1.

We want to prove that at iteration $k = n+1$, the representation $x_q^{(n+1)}(s,o)$ aggregates all path representations up to a length $n+1$ from the source. This is achieved by aggregating all edges that contain the nodes in the set:

$$U_{s,o}^{(0,n)} = \{u \mid (u,r,o) \in \mathcal{E}_o,\ u \in \{V_s^0, V_s^1, \cdots, V_s^n\}\}. \tag{D.11}$$

Per the previous inductive step, we assumed that the representations $x_q^n(s,o)\ \forall o \in V_s^n$ contain all path representations up to length n. Furthermore we noted that at iteration $k = n$, the representations for each node in the set $U_{s,o}^{(0,n-1)}$ must also contain all path representations up to a length $n-1$. Since $U_{s,o}^{(0,n)} = U_{s,o}^{(0,n-1)} \cup V_s^n$, this implies that $U_{s,o}^{(0,n)}$ contain all path representations up to length n. Thereby when $x_q^{(n+1)}(s,o)$ aggregates the nodes in $U_{s,o}^{(0,n)}$ it aggregates all path representations up to a length $n+1$. A node $o \in V_s^1$ will aggregate such nodes at iteration $k = n+1$ per both constraints.

This proves by induction that for $o \in V_s^1$, their representation $x_q^{(1+\delta)}(s,o)$ aggregates all path representations of length less than or equal to $1+\delta$.

D.2 Further Details on TAGNet

D.2.1 TAGNet Algorithm

The algorithm for TAGNet, with a fixed δ, is presented in Algorithm D.1.

D.2.2 Time Complexity Analysis

Per the constraints in Eq. (5.6), each node can be updated at most $\delta+1$ times and each edge can be aggregated at most $\delta+1$ times. The shortest path distance from a source node s to all other nodes can be calculated in linear time via a breadth-first search. The worst-case complexity for the standard version of TAGNet is therefore:

$$O\left((\delta+1)\cdot\left(|V|d^2 + |E|d\right)\right). \tag{D.12}$$

Of note is that the worst case-complexity is independent of the number of layers. This allows for much deeper propagation.

Algorithm D.1 TAGNet Algorithm (fixed δ)

Require:

s = Source node
q = Query relation
T = Max Number of Layers
$\mathbf{x}$ = Embeddings
δ = Offset
Agg-Degree = Whether to include degree msgs

Initialize:
  $x^{(0)}_{(s,o)} = \mathbf{0},\ \forall o \in \mathcal{V}$
  $x^{(0)}_{(s,o)} = \mathbf{x}_q$
for $t = 1...T$ **do**
  for $o \in \mathcal{V}$ **do**
    if $t - \delta \leq \text{dist}(s, o) \leq t$ **then**
      $C(s, o, t) = \{(u, r, o) \in \mathcal{E}(o) \mid \text{dist}(s, u) < \text{dist}(s, o) + \delta\}$
      $\text{Msgs} = \{\mathbf{x}^{(t-1)}_{(s,u)} \odot \mathbf{x}^{(t)}_r \mid (u, r, o) \in C(s, o, t)\}$
      if Agg-Degree **then**
        $\rho_o = b_o - |\text{Msgs}|$
        $\text{Msgs} = \text{Msgs} \cup \left\{\rho_v \cdot \mathbf{x}^{(t)}_{\text{deg}}\right\}$
      end if
      $\mathbf{x}^{(t)}_{(s,o)} = \text{Aggregate}\{\ \text{Msgs}\ \}$
    end if
  end for
end for
return $\mathbf{x}^{(\text{dist}(s,o)+\delta)}_{(s,o)}$ for all $o \in \mathcal{V}$

We further discuss the complexity when utilizing degree messages and a target-specific δ. As noted in Section 5.3.3, the inclusion of degree messages is equivalent to aggregating an additional edge each iteration. As such, it doesn't effect the model complexity. Furthermore, when utilizing a target-specific δ, an additional $(\delta + 1) \cdot d^2$ operations are added to calculate the attention scores. This is equivalent to updating each one node one additional time and therefore also has no effect on the model complexity.

D.2.3 TAGNet + A*Net

We further experiment with combining the pruning strategy of both A*Net and TAGNet. This is achieved by taking the intersection of the edge sets produced by both methods for a node pair (s, o) at iteration t. This is because we only want to aggregate an edge if it is not pruned by both

methods. For TAGNet, the edge set $C(s, o, t)$ is defined as in Eq. (5.6). We further denote the edge set for A*Net as $\mathcal{A}(s, o, t)$. Adapting Eq. (5.7) we arrive at:

$$\mathbf{x}_q^{(t)}(s, o) = \left(\bigoplus_{(v,r,o) \in C(s,o,t) \cap \mathcal{A}(s,o,t)} \mathbf{x}_q^{(t-1)}(s, v) \otimes \mathbf{w}_q(v, r, o) \right) \oplus \mathbf{x}_q^{(0)}(s, o). \tag{D.13}$$

The performance and efficiency when combining both methods is detailed in Section 5.4.1 and 5.4.2, respectively. Lastly, we note that we don't consider combining with the pruning strategy in AdaProp [144] due to its strong similarity with that of A*Net.

D.3 Experimental Settings

D.3.1 Datasets

We conduct experiments on both the transductive and inductive settings. For the transductive setting, we consider FB15K-237 [111] and WN18RR [30]. For the inductive setting, where the train and test entities are disjoint, we consider the splits generated by [109] from both FB15K-237 and WN18RR. Of note is that we omit the NELL-995 [126] dataset from both sets of our experiments. This is due to concerns raised by [98], where they argue that most of the triples in NELL-995 are either meaningless or trivial. The statistics for all the transductive and inductive datasets are given in Tables D.1 and D.2, respectively.

Table D.1 Statistics for Transductive Datasets.

Statistic	**FB15K-237**	**WN18RR**
#Entities	14,541	40,943
#Relations	237	11
#Train	272,115	86,835
#Validation	17,535	3,034
#Test	20,466	3,134

D.3.2 Baselines

In the transductive setting, following [154], we consider a variety of different models. For embedding-based methods we consider TransE [14] (performance from [82]), DistMult [127], ComlEx [112]. For GNN methods we include R-GCN [100] (performance on WN18RR taken from [154]) and CompGCN [113]. For path-based methods we include DRUM [96], NBFNet [154],

Table D.2 Statistics for Inductive Datasets.

Dataset		#Relations	Train			Validation			Test		
			#Entities	#Query	#Fact	#Entities	#Query	#Fact	#Entities	#Query	#Fact
FB15k-237	v1	180	1,594	4,245	4,245	1,594	489	4,245	1,093	205	1,993
	v2	200	2,608	9,739	9,739	2,608	1,166	9,739	1,660	478	4,145
	v3	215	3,668	17,986	17,986	3,668	2,194	17,986	2,501	865	7,406
	v4	219	4,707	27,203	27,203	4,707	3,352	27,203	3,051	1,424	11,714
WN18RR	v1	9	2,746	5,410	5,410	2,746	630	5,410	922	188	1,618
	v2	10	6,954	15,262	15,262	6,954	1,838	15,262	2,757	441	4,011
	v3	11	12,078	25,901	25,901	12,078	3,097	25,901	5,084	605	6,327
	v4	9	3,861	7,940	7,940	3,861	934	7,940	7,084	1,429	12,334

RED-GNN [143], A*Net [152], and AdaProp [144]. We note that for AdaProp the original results from [144] utilize 7 and 8 layers for FB15k237 and WN18RR, respectively (see Table 7 in [144]). For other methods such as TAGNet, NBFNet, and A*NET, the number of layers is fixed at 6. To facilitate a fair comparison, we run AdaProp on both datasets using 6 layers. We utilize the official source code [1] and the published hyperparameters.

For the inductive setting, following [109, 154], we include GraIL [109], CoMPILE [70], and NeuralLP [128] in addition to NBFNet and A*Net. We note that embedding methods aren't applicable to the inductive setting as the train and test entities are disjoint. For NBFNet, the results on the inductive FB15k-237 splits are reported by us while the results for the WN18RR splits are from [152]. This is because we observed that we can achieve better performance for NBFNet on the FB15k-237 splits than what was reported in [152]. Lastly, as with the transductive setting, we run AdaProp with 6 layers to facilitate a fair comparison between it and other path-based GNNs. We also set the hidden dimension to 32 as is with all other path-based GNNs.

D.3.3 Evaluation Metrics

In the transductive setting, we report the mean reciprocal rank (MRR), Hits@1, and Hits@10 following the filtered setting as described in [14]. For the inductive setting, following [144, 152], we only report the Hits@10.

D.3.4 Hyperparameter Settings

We list the parameters settings for TAGNet. Under the fixed-δ formulation it is trained for 20 and 16 epochs on the transductive and inductive setting, respectively. For the specific-δ formulation,

[1] https://github.com/LARS-research/AdaProp

we train for 25 and 20 epochs on the transductive and inductive setting, respectively, as we've found it takes longer to converge. For all transductive and inductive experiments in Table 5.1 and 5.2 we set the maximum number of layers to 6 and the hidden dimension to 32. This is to facilitate a fair comparison with NBFNet and A*Net. Furthermore the transductive batch size is fixed at 16. The number of negative samples is tuned from $\{128, 512, 1024, 2048\}$, the dropout from the range $[0, 0.7]$, the learning rate decay from $\{0.9, 0.95, 1\}$, the weight decay from [1e-8, 1e-3], and the adversarial temperature from $\{0.5, 1\}$. For the target specific setting we further test on setting g as its own function or as equal to the score function, $g = f$. We further tune the softmax temperature for attention from $\{0.5, 1, 5\}$. For the inductive setting we further tune the batch size from $\{16, 32, 64, 128\}$ and the learning rate from [1e-4, 1e-2]. Lastly, for all experiments, the offset δ is tuned from $\{1, 2, 3\}$.

D.3.5 Implementation Details

The framework is implemented with PyTorch [86]. All experiments were run on a single 32G Tesla V100 GPU. We train TAGNet with the binary cross-entropy loss optimized via the Adam optimizer [52]. We follow [128] and augment the graph by including reciprocal edges, such that for an edge (h, r, t), its reciprocal edge (t, r^{-1}, h) is included. In this scenario r^{-1} is considered a distinct relation from r.

D.4 Additional Analysis on TAGNet

In this section we take a closer look as to what kind of messages are pruned by TAGNet. As noted in Section 5.3.1 we strive to limit the number of empty and redundant messages. We first analyze how well TAGNet can prune both of those messages. We then examine the reason why some datasets may prune more empty or redundant messages.

We first analyze the number of **empty** and **redundant** messages pruned for both transductive datasets. We report the results in Table D.3 as a % of the total number of pruned messages. E.g., For FB15k-237 51% of the total number of pruned messages are empty messages. For simplicity, we limit this study to the the best versions of each model, i.e. $\delta = 2$ for FB15K-237 and $\delta = 3$ for WN18RR. We find that on FB15k-237, the messages pruned are split evenly between empty and

redundant messages. On the other hand, for WN18RR over 90% of the messages pruned are empty messages.

Table D.3 % of Messages Pruned that are either Empty or Redundant.

Dataset	**% Empty**	**% Redundant**
FB15k-237	51%	49%
WN18RR	91%	9%

An obvious question is: *Why does the composition of pruned messages differ between datasets?* We believe this can be explained via two properties of each datasets, the density and distance distribution. We measure the sparsity via the mean degree, which is shown in Table D.4. We do this as graphs with a low mean degree will contain few connections between nodes, resulting in fewer paths between different nodes and thereby fewer redundant paths. Furthermore, there will be a lower chance of a node visiting another node already on the path, as most nodes are linked to only a handful of nodes. We further show the distance distribution of the test samples, i.e., the % of test samples that are a distance k from each other, in Table D.5. This is because when nodes are typically far from each other, the target nodes will aggregate many empty messages. Using Figure 5.1a as an example, the source and node 7 are a distance 3 from each other. Because of this, in the first two iterations NBFNet will propagate node 6 to node 7, even though node 6 contains no information. However, this is less of an issue between nodes of shorter distances as there fewer iterations needed to reach it. From this, we hypothesize that graphs that feature, on average, a larger distance between nodes will propagate more empty messages.

Table D.4 Mean Degree of Transductive Datasets.

Dataset	**Mean Degree**
FB15k-237	18.7
WN18RR	2.1

From the results in Table D.4 and D.5 we make the following observations: **(a)** WN18RR is much sparser than FB15k-237. The higher density of FB15k-237 leads to many more paths and subsequent opportunities to visit a node already on the path. The opposite is true for WN18RR as

Table D.5 Distance Distribution of Test Samples on the Transductive Datasets.

Distance	FB15k-237	WN18RR
1	0%	35%
2	73%	9%
3	26%	21%
4	0.2%	7%
5	0.005%	9%
6+	0%	18%

since the average degree is low, few paths exist in the graph. This results in many more redundant paths existing in FB15k-237 as compared to WN18RR. **(b)** For FB15k-237, the vast majority of test samples are close to each other. This leads to less empty messages. However, for WN18RR the distance covers a much wider range. For example, over 33% of test samples have a distance of 4+ between them. This is only true for 0.205% of samples on FB15k-237. This helps explain why TAGNet mostly prunes empty messages on WN18RR, as the larger distance between nodes leads to many messages that contain no information.