Bring Your Own View: Graph Neural Networks for Link Prediction with Personalized Subgraph Selection

Qiaoyu Tan
Texas A&M University
qytan@tamu.edu

Xin Zhang
The Hong Kong Polytechnic University
xin12.zhang@connect.polyu.hk

Ninghao Liu
University of Georgia
ninghao.liu@uga.edu

Daochen Zha
Rice University
daochen.zha@rice.edu

Li Li
Samsung Electronics America
li.li1@samsung.com

Rui Chen
Samsung Electronics America
rui.chen1@samsung.com

Soo-Hyun Choi*
Samsung Electronics
sh9.choi@samsung.com

Xia Hu
Rice University
xia.hu@rice.edu

ABSTRACT

Graph neural networks (GNNs) have received remarkable success in link prediction (GNNLP) tasks. Existing efforts first predefine the subgraph for the whole dataset and then apply GNNs to encode edge representations by leveraging the neighborhood structure induced by the fixed subgraph. The prominence of GNNLP methods significantly relies on the adhoc subgraph. Since node connectivity in real-world graphs is complex, one shared subgraph is limited for all edges. Thus, the choices of subgraphs should be personalized to different edges. However, performing personalized subgraph selection is nontrivial since the potential selection space grows exponentially to the scale of edges. Besides, the inference edges are not available during training in link prediction scenarios, so the selection process needs to be inductive. To bridge the gap, we introduce a Personalized Subgraph Selector (PS2) as a plug-and-play framework to automatically, personally, and inductively identify optimal subgraphs for different edges when performing GNNLP. PS2 is instantiated as a bi-level optimization problem that can be efficiently solved differently. Coupling GNNLP models with PS2, we suggest a brand-new angle towards GNNLP training: by first identifying the optimal subgraphs for edges; and then focusing on training the inference model by using the sampled subgraphs. Comprehensive experiments endorse the effectiveness of our proposed method across various GNNLP backbones (GCN, GraphSage, NGCF, LightGCN, and SEAL) and diverse benchmarks (Planetoid, OGB, and Recommendation datasets). Our code is publicly available at https://github.com/qiaoyu-tan/PS2

*Corresponding author.

CCS CONCEPTS

KEYWORDS

ACM Reference Format:

1 INTRODUCTION

Graph is a ubiquitous and powerful data structure to present different types of relational data, such as social networks and biological molecules. Given that real-world graphs are often only partially observed, *link prediction* [1], which aims to predict missing links in a graph, is a central problem across many scientific domains. For example, link prediction has applications in predicting protein interactions [2], drug responses [3], and completing the knowledge graph [4, 5]. Besides, it is also the backbone for various recommendation systems, e.g., friend suggestion in social networks [6, 7] or product recommendation in online market-places [8–10].

Recently, considerable efforts have been made to develop advanced link prediction techniques [11–13]. Among them, graph neural networks (GNNs) based link prediction models (GNNLP) have achieved impressive results [14–20], owing to the expressive encoding capacity of GNNs. The essential idea behind GNNLP is to generate edge representation based on the subgraph around the anchor edge via a GNN encoder and then estimate its likelihood with a prediction function. According to the difference in utilizing subgraph for edge embedding, they can be divided into two categories: node2link [14, 15, 17] and subgraph2link [18, 19, 21]. The node2link approaches (i.e., GAE [14], GraphSage [15], and Light-GCN [17]) aim to first learn node representations for the head and

tail nodes of the anchor edge independently, and then combine the representations of end nodes for edge embedding. In contrast, the subgraph2link approaches (e.g., SEAL [18] and [19]) target to learn edge representation by pooling over the subgraph of the given edge, casting it as a graph representation learning task.

While effective, both of them have largely overlooked the diversity of subgraphs when embedding different edges. For example, GAE and SEAL assume that the best subgraph structure for all edges is the same and adopt the neighbors within k-hops for edge embedding, where $k \in \{1, 2, \cdots, K\}$ is a hyperparameter. Although such collective selection can significantly reduce the tedious tuning efforts to identify the best k value from K options, the shared subgraph structure assumption is rather limited. Different edges may favor different subgraph structures for link prediction. This hypothesis is reasonable because node connectivity patterns in real-world graphs are complex [22, 23]. For instance, in social networks, the social connectivity of users is created by different factors [24, 25]. In the recommendation system, a user's purchase behavior could be motivated by either his/her like-minded customers or conceptually similar products in the historical records [26].

Motivated by this, we conduct a preliminary experiment on the Cora dataset to test how different subgraph structures impact the link prediction results in Figure 1. We observe that GAE can accurately infer different missing edges by training over various predefined subgraph structures. For instance, the first missing edge can be well recovered by using the neighbors within 3 hops, while the second missing edge can be effectively reconstructed by using the 2-hops and 3-hops of neighbors of its head and tail nodes, respectively. In terms of their input neighborhood subgraphs, this personalized phenomenon of edges, has never been explored in link prediction scenarios. To bridge the gap, in this paper, we propose to develop an effective subgraph selector to automatically identify the most informative subgraphs for different edges.

However, it is a nontrivial and challenging task mainly because of three roadblocks. First, given a graph data, the latent subgraph selection space is exponential to the size of edges, which is millions or even billions in practice. It is impossible to identify the optimal subgraph configurations for all edges via the brute-force search. Second, in link prediction applications, the edges to be predicted are not available during the training. Thus, the subgraph selection process must be inductive, enabling infer subgraph structures for unseen edges. Third, how to make the edge-wise subgraph selection adaptive to the well-established base models such as methods under the node2link approach (e.g., GAE [14], GraphSage [15], NGCF [16] and LightGCN [17]) or subgraph2link category (e.g., SEAL [18]).

To address these challenges, we propose a novel personalized subgraph selector, dubbed PS2, as a plug-and-play framework. It aims to develop an automatic and inductive subgraph selection module for GNNLP methods, such that the most informative subgraph structures can be explicitly identified and exploited for different edges. Specifically, we aim to explore two important research questions. (i) How to automatically sample the optimal subgraph structure for each edge efficiently, and make the selection process inductive? (ii) How to effectively equip well-established GNNLP methods with the proposed personalized selector, so as to offer orthogonal gains across a variety of graph domains and GNNLP backbones? We summarize our major contributions as follows.

- We focus on subgraph selection for GNNs based link prediction (GNNLP) problem, and propose an effective personalized subgraph selector (PS2). PS2 is the first to automate subgraph selection in an edge-wise fashion when performing GNLLP. It can be easily adopted to boost the well-studied GNNLP methods.
- PS2 can be formulated under bi-level optimization and solved using the alternating gradient-descent algorithm. It is inspired by the differentiable architecture search [27], but we extend it from a transductive search model to an inductive subgraph selector, focusing on the edge-wise subgraph structure selection rather than the model architecture search.
- We conduct extensive experiments to evaluate PS2 on multiple graph benchmarks of diverse types and scales, over a variety of GNNLP backbones. Empirical results show that with PS2, the performance of state-of-the-art GNNLP competitors can be advanced with a wide margin.
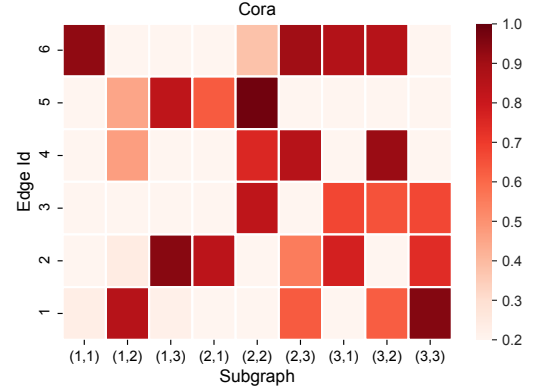

Figure 1: The effect of different edge subgraphs towards 6 randomly sampled test edges on the Cora dataset. The results are obtained by training GAE [14] nine times with different subgraph ranges. For example, $(2, 3)$ means the subgraph of an edge is composed of the 2-hops and 3-hops of neighbors of its head and tail nodes, respectively. The X-axis denotes different subgraphs selection strategies, and the Y-axis is the id of the sampled edges. The color from light to dark represents the predicted probability for edge existence.

2　PRELIMINARIES

Problem Formulation. We are given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with n nodes, where $\mathcal{V}$ and $\mathcal{E}$ denote the sets of nodes and edges, respectively. We use $\mathcal{G}_v^k$ to represent the local subgraph of node $v \in \mathcal{V}$, and $\mathcal{G}_{u,v}^k = (\mathcal{G}_u^k, \mathcal{G}_v^k)$ denotes the subgraph of edge $e_{u,v} \in \mathcal{E}$ within k hops, in which k is a crucial hyperparameter. Note that $\mathcal{G}_{u,v}^k$ is obtained by removing the redundancies (e.g., repeated edges and nodes) in subgraphs $\mathcal{G}_u^k$ and $\mathcal{G}_v^k$. In previous studies, the optimal k value is manually picked from the set $\{1, 2, \cdots, K\}$ via either rule of thumb or validation, where the same k is applied to the whole graph $\mathcal{G}$. However, as shown in Figure 1, the optimal k varies for predicting different edges. Therefore, we propose a personalized subgraph selection to identify the most informative subgraph for different edges, where the problem is formally defined below.

Definition 1. ***Personalized subgraph selection.*** *Given a graph* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, *the subgraph space for node* $v \in \mathcal{V}$ *is defined as*
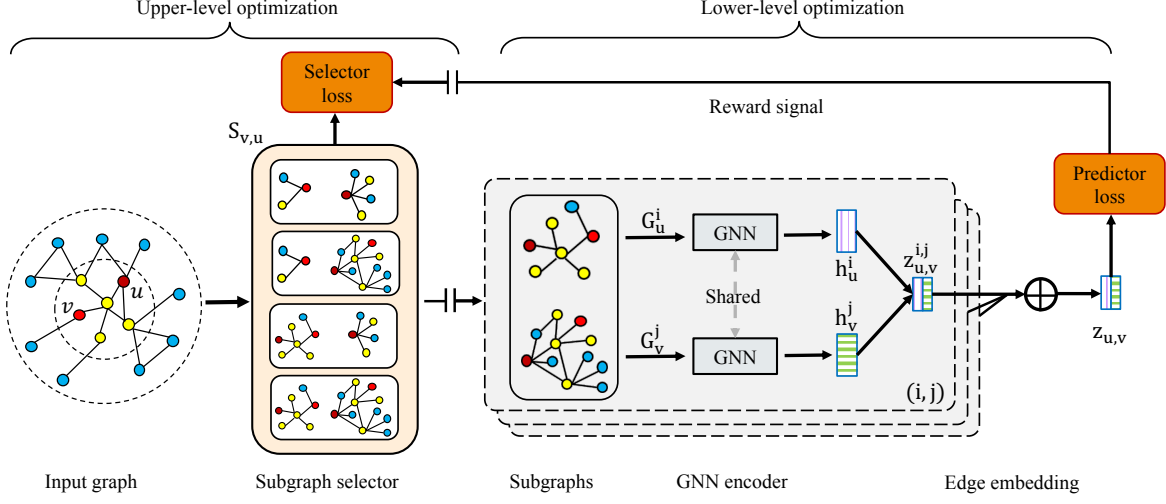
Figure 2: The training pipeline (i.e., search phase) of the proposed PS2. The subgraph selector generates an importance score for each subgraph candidate. The upper-level optimization updates the parameters of the selector g_θ. The lower-level optimization updates the parameters of the GNN f_w and predictor q_w.

$\mathcal{S}_v = \{\mathcal{G}_v^1, \mathcal{G}_v^2, \cdots, \mathcal{G}_v^k, \cdots, \mathcal{G}_v^K\}$. *To predict edge $e_{u,v}$, the personalized subgraph selection aims to find the optimal subgraph $\mathcal{G}_{u,v} = (\mathcal{G}_u^i, \mathcal{G}_v^j | \mathcal{G}_u^i \in \mathcal{S}_u, \mathcal{G}_v^j \in \mathcal{S}_v)$. The values of i and j are determined adaptively for different target edges.*

Compared with existing GNNLP methods, our personalized setting implies two essential properties as below.

- **Edge subgraph is personal.** In our setting, the local subgraphs are personalized to different edges. For example, the subgraph can be $\mathcal{G}_{1,2} = (\mathcal{G}_1^1, \mathcal{G}_2^3)$ in predicting edge $e_{1,2}$, but it also can be $\mathcal{G}_{3,4} = (\mathcal{G}_3^2, \mathcal{G}_4^1)$ for edge $e_{3,4}$. However, in existing GNNLP efforts, the subgraph order is restricted to be the same for all edges, i.e., $\mathcal{G}_{1,2} = (\mathcal{G}_1^2, \mathcal{G}_2^2)$ and $\mathcal{G}_{3,4} = (\mathcal{G}_3^2, \mathcal{G}_4^2)$.
- **Node subgraph is polysemous.** Node v can use different subgraphs in predicting different edges. For example, the optimal subgraph for $e_{1,2}$ is $\mathcal{G}_{1,2} = (\mathcal{G}_1^1, \mathcal{G}_2^3)$, while $\mathcal{G}_{1,4} = (\mathcal{G}_1^3, \mathcal{G}_4^2)$ for edge $e_{1,4}$. In existing GNNLP methods, the neighbor range of node subgraph is fixed, e.g., subgraphs for node 1, 2, and 4 are $\mathcal{G}_1^2, \mathcal{G}_2^2$, and $\mathcal{G}_4^2$, respectively.

GNNs for Node Embedding. Given a node $v \in \mathcal{V}$, GNNs models [28, 29] are widely adapted to mapping nodes into hidden representations, i.e., $f_w : \mathcal{V} \to \mathbb{R}^d$. GNNs target to update the node presentation by aggregating representations of itself and its neighbors. Formally, at the k-th layer, we have

$$\mathbf{h}_v^{(k)} = \text{UPDATE}(\mathbf{h}_v^{(k-1)}, \text{AGGREGATE}(\{\mathbf{h}_u^{(k-1)} : u \in \mathcal{N}_v\})). \quad (1)$$

$\mathbf{h}_v^k \in \mathbb{R}^d$ is the hidden representation of node v at the k-th layer, while $\mathcal{N}_v$ is the set of nodes adjacent to v. We often initialize $\mathbf{h}_v^0$ as $\mathbf{x}_v$. The AGGREGATE function aims to receive messages from neighbors and the UPDATE function focuses on updating v's representation based on the representation from the previous GNN layer and the information from neighbors. By stacking K GNN layers, each node v has K hidden representations $\{\mathbf{h}_v^1, \cdots, \mathbf{h}_v^K\}$.

3 THE PROPOSED METHOD

In this section, we present the details of the proposed PS2 shown in Figure 2. We first discuss the exponential subgraph selection space of our problem. Then, we elaborate on a tailored inductive subgraph selector to effectively sample subgraphs for seen and unseen edges in this space. Finally, we show how to formulate our training objectives into bi-level optimization and solve it via alternating gradient descent.

3.1 Subgraph Selection Space

Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and the maximum number of hops K considered, there are K latent subgraphs for each node $v \in \mathcal{V}$, denoted by $\mathcal{S}_v = \{\mathcal{G}_v^k\}_{k=1}^K$. Each subgraph $\mathcal{G}_v^k$ is spanned by the anchor node v and its neighbors within k hops, whose shortest path distance from v is less than k. In previous studies, k is a dataset-level hyperparameter, which is fixed for all nodes/edges in the graph $\mathcal{G}$. In this work, we denote the subgraph of edge $e_{u,v}$ as $\mathcal{G}_{u,v}^k = (\mathcal{G}_u^k, \mathcal{G}_v^k)$. In practice, the best k value is usually selected through validation and is applied to the whole graph. This collective selection strategy has been widely adopted as the default protocol in prior GNNLPs.

However, as discussed before, applying a constant k value in selecting subgraphs leads to suboptimal results for predicting some edges. Thus, we propose to adaptively choose the subgraph selection space for different edges. Specifically, we define the subgraph of edge $e_{u,v}$ to be composed of its end nodes' subgraphs. That is, $\mathcal{G}_{u,v} = (\mathcal{G}_u^i, \mathcal{G}_v^j | 1 \le i, j \le K)$. Then, the potential subgraph pool size for each edge is K^2, and the total subgraph selection space equals to $K^{2|\mathcal{E}|}$ for the whole graph, where $|\mathcal{E}|$ is the number of edges in $\mathcal{G}$. Although K is empirically small (e.g., $K = 2, 3$) in link prediction scenarios, the selection space in our personalized setting is still huge and intractable as **the complexity grows exponentially with the edge size**. For example, when $K = 2$ and $|\mathcal{E}| = 50$, we have nearly 10^{30} selection candidates. The situation is more difficult in real-world graphs, where $|\mathcal{E}|$ is millions or even billions.

In summary, by personalizing edge subgraphs, the subgraph selection space for link prediction increases from K to $K^{2|\mathcal{E}|}$. Therefore, existing strategies based on the rule of thumb or grid search are no longer appropriate. Also, a tailored subgraph space selector is needed to tackle our personalized subgraph selection problem.

3.2 Personalized Subgraph Selector

To assign different subgraph orders to different edges for link prediction, the intuitive solution is random selection. For example, given an edge $e_{u,v}$, we can randomly select its subgraph (e.g., $\mathcal{G}_{u,v} = (\mathcal{G}_u^1, \mathcal{G}_v^3)$) from the candidate pool. Despite the simplicity, the random selection approach fails to control the quality of the resulting subgraphs. Coupling existing GNNLP methods with random subgraph selection could incur significant performance degradation, especially when the graph is challenging, e.g., on OGB datasets (see Table 3).

To address this issue, we focus on data-driven selection by making the subgraph selection process learnable. The core idea is to parameterize the subgraph selector with a deep neural network, which takes a query edge $e_{u,v}$ as input and outputs its optimal subgraph for downstream link prediction. The main challenges to achieving this goal are two-fold. (i) Given the exponential complexity (i.e., $K^{|\mathcal{E}|}$) of the selection space, how can we make the personalized subgraph selection scale to real-world graphs with millions or even billions of edges? (ii) Since the edges to be inferred are not available in training under link prediction scenarios, how to make the subgraph selector inductive to unseen edges? We introduce our solutions below.

3.2.1 Learnable Subgraph Selector.
To tackle the first challenge, we propose to make the selection process learnable. Given an edge $e_{u,v}$ and its K^2 subgraph candidates, i.e., $\{(\mathcal{G}_u^i, \mathcal{G}_v^j)\}_{i,j=1}^K$, our subgraph selector aims to find the most informative subgraph from the candidate set. This selection problem is well-known to be discrete and non-differentiable. While enormous efforts based on evolution [30] or reinforcement learning [31] have been dedicated to addressing the discrete selection problem, they are inefficient for training. To tackle this problem, we make the selection process learnable by relaxing the discrete selection space to be continuous. The core idea is to relax the selection of a single subgraph to a softmax over all possible subgraph candidates. Formally, let $\alpha_{u,v}^{i,j}$ denote the contribution of subgraph $(\mathcal{G}_u^i, \mathcal{G}_v^j)$ in predicting edge $e_{u,v}$, the learnable selection process is defined as:

$$\mathcal{G}_{u,v} = \sum_{1 \le i,j \le K} \frac{\exp(\alpha_{u,v}^{i,j}/\tau)}{\sum_{i',j'=1}^K \exp(\alpha_{u,v}^{i',j'}/\tau)} * \mathcal{G}_{u,v}^{i,j}, \qquad (2)$$

we use $*$ to denote the multiplication alike operation on the subgraph in theory. $\alpha_{u,v} = [\alpha_{u,v}^{1,1}, \cdots, \alpha_{u,v}^{K,K}] \in \mathbb{R}^{K^2}$ is the subgraph related weight vector for edge $e_{u,v}$, which is initialized as part of model parameters. τ is a small temperature parameter, which helps approximate the categorical selection distribution. By Eq. (2), the subgraph selection process reduces to learning a set of continuous variables $\{\alpha_{u,v} : e_{u,v} \in \mathcal{E}\}$. After the subgraph selector is well-trained, a discrete subgraph selection can be acquired by replacing

the mixed selection with the most likely subgraph, i.e., $\mathcal{G}_{u,v} = \mathcal{G}_{u,v}^{i,j}$ if $\alpha_{u,v}^{i,j} = \arg\max_{i',j'} \alpha_{u,v}^{i',j'}$.

Although Eq. (2) makes our personalized subgraph selection learnable, it still cannot resolve the second challenge. This is because it only learns weight vectors $\{\alpha_{u,v} : e_{u,v} \in \mathcal{E}\}$ for observed edges, yet cannot generate weight variables for unseen ones. As a result, it cannot be applied to infer missing edges.

3.2.2 Inductive Subgraph Selector.
To address the second challenge, we propose to make Eq. (2) inductive by computing the weight vector α_v with a deep neural network. Specifically, we estimate the contribution score $\alpha_{u,v}^{i,j}$ of edge $e_{u,v}$ w.r.t. the subgraph $\mathcal{G}_{u,v}^{i,j}$ via a score function $g_\theta : \mathcal{G}_{u,v} \to R$. Here g_θ takes edge subgraphs as input and outputs their selection scores. Following this principle, we rewrite Eq. 2 into an inductive version:

$$\mathcal{G}_{u,v} = \sum_{1 \le i,j \le K} \frac{\exp(g_\theta(\mathcal{G}_{u,v}^{i,j})/\tau)}{\sum_{i',j'=1}^K \exp(g_\theta(\mathcal{G}_{u,v}^{i',j'})/\tau)} * \mathcal{G}_{u,v}^{i,j}. \qquad (3)$$

The above equation provides a principled solution to our personalized subgraph problem. On the one hand, it allows efficient subgraph selection for different edges based on the simple forward pass of a neural network. On the other hand, it enables the selection of the most informative subgraphs for unseen edges based on their subgraph characteristics. We now illustrate how to implement the score function g_θ. Since g_θ takes K^2 subgraphs $\{\mathcal{G}_{u,v}^{i,j} : 1 \le i, j \le K\}$ of an edge $e_{u,v}$ as input, an intuitive strategy is applying GNNs to encode the K^2 subgraphs independently. However, this schema is time-consuming in training because it requires roughly K^2 GNNs forward passes to iterate over all edges once.

Efficient K^2 subgraphs embedding. To avoid running GNNs forward pass repeatedly, we propose to get the representations of K^2 edge subgraphs simultaneously, by directly combining the hidden representations of end nodes from K GNN layers. Specifically, given the K hidden representations of node u $\{\mathbf{h}_u^k : 1 \le k \le K\}$ and v $\{\mathbf{h}_v^k : 1 \le k \le K\}$, we generate the representation of edge $e_{u,v}$ in terms of subgraph $(\mathcal{G}_u^i, \mathcal{G}_v^j)$ as $\mathbf{z}_{u,v}^{i,j} = \text{COM}(\mathbf{h}_u^i, \mathbf{h}_v^j)$. $\text{COM}(\cdot, \cdot)$ is a combination function, and the default setting is the element-wise multiplication. This approximation is reasonable in GNNs since $\mathbf{h}_v^i$ is obtained by aggregating messages from v's neighbors within i hops. **By doing this, we don't need to extract k^2 subgraphs and apply GNN on these graphs separately. Instead, we can directly obtain K^2 subgraph embeddings upon the hidden representations of one GNN forward. So, the computational complexity of plugging in our personalized selector is close to standard GNNLPs (See Section 5.6 for efficiency analysis).**

After obtaining K^2 edge representations, we feed them into a MLP layer with ReLU activation function to predict their importance scores, i.e., $\alpha_{u,v}^{i,j} = g_\theta(\mathbf{z}_{u,v}^{i,j})$. Through relaxing the hard selection operation, we can rewrite the mixed selection process in Eq. (3) in embedding space as:

$$\mathbf{z}_{u,v} = \sum_{1 \le i,j \le K} \frac{\exp(g_\theta(\mathbf{z}_{u,v}^{i,j})/\tau)}{\sum_{i',j'=1}^K \exp(g_\theta(\mathbf{z}_{u,v}^{i',j'})/\tau)} \mathbf{z}_{u,v}^{i,j}. \qquad (4)$$

$\mathbf{z}_{u,v} \in \mathbb{R}^d$ is the final representation of edge $e_{u,v}$. It is a mixed representation obtained by summing over the representations of

various subgraph forms. Note that the mixed operation is only applied for the search phase. In the application phase, we output one subgraph for each edge via max selection (see Section 4).

3.3 Model Training

After generating the edge representation $\mathbf{z}_{u,v}$, we adopt an edge-wise loss function to estimate the reconstruction errors, expressed as:

$$\mathcal{L} = - \sum_{(v,u) \in \mathcal{E}} \frac{\exp(y_{u,v})}{\sum_{u' \in \mathcal{V}} \exp(y_{v,u'})}, \qquad (5)$$

where $y_{u,v} = q_w(\mathbf{z}_{u,v})$ is the predicted score for edge $e_{u,v}$, and q_w is another multilayer perceptron with ReLU activation. As the sum operation in the denominator of Eq. (5) is computationally expensive, we adopt negative selection techniques [15] to accelerate the optimization in experiments.

In the search phase, our goal is to jointly learn the subgraph selector g_θ and the model weights within the mixed selection, including GNN encoder f_w and link predictor q_w. Following [27, 31], we employ the validation set performance as a reward to optimize the subgraph selector, and train the GNN encoder and predictor by fitting the training set. Specifically, we optimize our model via the following bi-level optimization framework:

$$\min_{g_\theta} \mathcal{L}_{valid}(w^*, \theta) \text{ s.t. } w^* = \arg\min_w \mathcal{L}_{train}(\theta^*, w). \qquad (6)$$

We use w to wrap up the parameters of GNN encoder f_w and link predictor q_w for simplicity. $\mathcal{L}_{train}$ and $\mathcal{L}_{valid}$ denote the loss function in Eq. (5) computed based on the training and validation sets, respectively. The upper-level objective $\mathcal{L}(w^*, \theta)$ aims to find θ that minimizes the validation rewards given the optimal w^*, and the lower-level objective $\mathcal{L}(\theta^*, w)$ targets to optimize w by minimizing the training loss with θ fixed.

It is worth noting that Eq. (6) only exploits the cheap signals from observed edges, without accessing downstream labeled data for evaluation. Therefore, the validation set used to train the selector can be easily constructed. Since a closed-form solution cannot be computed, we optimize Eq. (6) via alternating between the lower-level and the upper-level objectives as below.

3.3.1 Lower-level optimization. With θ fixed, the lower-level optimization w.r.t. w follows the conventional gradient descent procedure, represented as:

$$w' = w - \lambda \nabla_w \mathcal{L}_{train}(w, \theta^*), \qquad (7)$$

where $\lambda \in \mathcal{R}_{>0}$ is the learning rate. The converged solution is denoted as $w^*(\theta)$.

3.3.2 Upper-level optimization. With w fixed, the upper-level optimization updates θ according to the validation performance as:

$$\theta' = \theta - \lambda \nabla_\theta \mathcal{L}_{valid}(w^*(\theta), \theta). \qquad (8)$$

However, evaluating the gradient w.r.t. θ exactly is computationally prohibitive, since it requires solving for the optimal $w^*(\theta)$ whenever θ gets updated. To approximate the optimal solution $w^*(\theta)$, we propose to take one step of gradient descent update for w, without solving the lower-level optimization completely by training until convergence. The full derivation is delegated to Appendix C. Here, we directly present the final result:

$$\nabla_\theta \mathcal{L}_{valid}(w^*(\theta), \theta) \approx \nabla_\theta \mathcal{L}_{valid}(w', \theta)$$
$$- \lambda \frac{\nabla_\theta \mathcal{L}_{train}(w^+, \theta) - \nabla_\theta \mathcal{L}_{train}(w^-, \theta)}{2\epsilon}, \qquad (9)$$

where $w^\pm = w \pm \epsilon \nabla_{w'} \mathcal{L}_{valid}(w'(\theta), \theta)$, and ϵ is a small scalar for finite difference approximation. By alternating between the two update rules in Eq. (7) and Eq. (8), we can learn an effective personalized subgraph selector that generalizes well for unseen edges. Although an optimizer with the theoretical guarantee of convergence for the bi-level optimization problem in Eq. (6) remains an open challenge, alternating gradient descent algorithm has been widely adopted to solve similar objectives in Bayesian optimization [32], automatic differentiation [33], and adversarial training [34]. Algorithm (1) in Appendix depicts the optimization procedure of our model. It shows some level of empirical convergence as seen in Figure 8 of Appendix.

4 APPLICATION PHASE

After the search phase, we can apply the *selected subgraphs* of different edges to various GNNLP models. In this section, we elaborate on two scenarios as examples. First, we illustrate how to train node2link-based models based on the selected edge subgraphs. Second, we show how to train subgraph2link-based methods given the sampled subgraphs.

4.1 Personalized node2link Based Models

Typical examples under the node2link approach include GAE, GraphSAGE, LightGCN, and NGCF, to name a few. Given the subgraph $\mathcal{G}_{u,v} = (\mathcal{G}_v^i, \mathcal{G}_u^j)$ of edge $e_{u,v}$, the models continue to learn node representations of end node subgraphs using GNNs, and then combine the representations of end nodes as the edge embedding towards prediction. In traditional settings, $i = j = k$, the edge representation can be easily generated by concatenating embeddings of end nodes in the last GNN layer, i.e., $\mathbf{z}_{u,v} = [\mathbf{h}_u^k, \mathbf{h}_v^k]$. When using personalized subgraph selection, where i and j could be different, we compute the personalized edge embedding via $\mathbf{z}_{u,v} = [\mathbf{h}_u^i, \mathbf{h}_v^j]$.
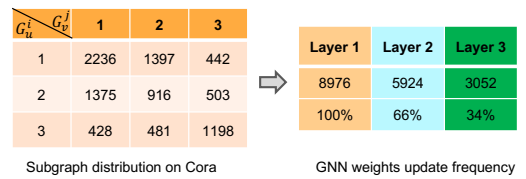

Figure 3: An example illustration of the subgraph imbalance issue. The frequency of updating three GNN layers differs due to the imbalanced edge distribution across layers.

4.1.1 Subgraph Imbalance Issue. However, directly training existing node2link-based models over the personalized edge subgraphs may suffer from the subgraph imbalance issue shown in Figure 3. We can observe that the three GNN layers will be updated inconsistently in mini-batch training, since the number of edges being encoded across three layers is different. To eliminate this issue, inspired by the success of the pre-training models in texts [35]

and images [36], we adopt the pre-train&finetune fashion to train node2link methods in personalized subgraph setting. By initializing the GNN encoder of the application model with the pre-trained one in the search phase, the application model can be well-tuned with limited training samples (a.k.a. limited training edges in the third layer). Note that the subgraph imbalance issue is different from class imbalance problem [37] in standard machine learning, since the training subgraphs in deeper layers are dependent on previous layers, which makes up-sampling or down-sampling techniques not applicable.

4.2 Personalized subgraph2link Based Models

Different from node2link-based methods, subgraph2link approaches naturally take the subgraphs of anchor edges as input, since they treat edge embedding as a graph-level representation learning task. Therefore, the personalized edge subgraphs generated by our model can be directly fed to them as input without additional effort.

5 EXPERIMENTS

We try to answer five research questions through experiments. **Q1:** Is considering personalized edge subgraphs beneficial for GNNLP models when evaluated on different applications? **Q2:** How effective is the proposed personalized subgraph selector in identifying edge subgraphs across various datasets? **Q3:** How will our personalized subgraph selector react to the changes in different optimization strategies? **Q4:** What are the impacts of hyperparameters on PS2, such as the maximum hop number K and the embedding dimension D of the score function? **Q5:** What is the running complexity of our personalized subgraph selector compared with standard GNNLPs?

5.1 Datasets and Experiment Settings

Datasets. For a comprehensive comparison, we use nine datasets of diverse nature with both homogeneous and heterogeneous graphs. For homogeneous graphs, we consider six popular datasets including three benchmark Planteoid datasets (**Cora**, **CiteSeer**, and **PubMed** [38]) and three large-scale benchmark datasets, **ogbl-ddi**, **ogbl-collab**, and **ogbl-ppa** from Open Graph Benchmark (OGB) [39]. We summarize their statistics in Table 4 of Appendix. For heterogeneous graphs, we include three benchmark datasets including **Gowalla** [40], **Yelp2018** [16], and **Amazon-book** [41]. We summarize their statistics in Table 5 of Appendix.

Learning protocols. We aim to provide a rigorous and fair comparison between different models across various graph domains by following the standard dataset splits and training procedure. For homogeneous graphs, we follow [14] to randomly split three graphs in Planetoid datasets into three sets, i.e., the training set (85%), the validation set (5%), and the test set (10%), and measure model performance based on AUC and Average Precision (AP) scores. For OGB datasets (ogbl-ddi, ogbl-collab, and ogbl-ppa), we follow [42] to split the datasets into three sets according to the split ratio summarized in Table 4, and evaluate the performance using Hit rate (Hit@N), where N is the number of nodes recalled. For heterogeneous graphs, we follow [17] to generate the training, validation, and testing sets with split ratios in Table 5. Since it is too time-consuming to rank all items for every user during evaluation, we follow the common strategy [43] that randomly samples 100 items that are not interacted

with by the user, ranking the test item among the sampled items. The performance of the ranked list is judged by two widely-used evaluation metrics: hit@N and ndcg@N.

Baselines. To demonstrate the effectiveness, we compare our model with state-of-the-art link prediction methods of two domains. For homogeneous graphs, we include two popular node2link based methods (**GAE** [14] and **GraphSage** [15]) and one subgraph2link based method (**SEAL** [18]). For heterogeneous graphs, we consider two recently proposed benchmark methods (**NGCF** [16] and **Light-GCN** [17]). Besides, we include one variant of our model based on the random search, named "RS". For all baseline methods, we use their open-source implementations with the best configurations on datasets that are tested in original papers. For datasets not originally tested, we tune their hyperparameters according to the range suggested in original papers.

Implementation details. Our model is built upon the Pytorch platform. We train our model for 100 epochs with Adam optimizer and early stopping with patience of 20 epochs. Following common practice in [14, 17, 42], we employ a three-layer GNN encoder with dimension 32, 256, and 64 for the Planetoid, OGB, and heterogeneous datasets, respectively. When applying our personalized subgraph selector PS2 to node2link-based baselines (GAE, GraphSage, NGCF, and LightGCN), we use the same GNN architectures as the vanilla counterparts in the search phase. For subgraph2link-based baseline (SEAL), we employ GCN [28] as the backbone in the search phase, since it is memory and time expensive to generate subgraph embeddings by pooling over the whole subgraph as SEAL does. Our model has two hyper-parameters, i.e., the maximum hop number K and the hidden dimension D of score function g_θ. We set $K = 3$ by default and search D within the set $\{64, 128, 256, 512, 1024\}$. The best options for three Planetoid and other datasets are 256 and 512, respectively. We provide more details in Appendix B.

5.2 Comparison with the Baselines

To answer the question **Q1**, we compare the performance of the proposed personalized subgraph selector with state-of-the-art baselines across homogeneous and heterogeneous domains. Table 1, Table 3 and Table 2 report the results over Planetoid, OGB, and three recommendation datasets, respectively. From the tables, we have the following **Obs**ervations.

Obs. 1. Through edge subgraph personalization, PS2 boosts the performance of link prediction across different domains. By comparing classical GNNLP methods (GAE, GraphSage, SEAL, NGCF, and LightGCN) with our personalized subgraph selector (GAE-PS2, GraphSage-PS2, SEAL-PS2, NGCF-PS2, and LightGCN-PS2), our model consistently outperforms the vanilla counterparts on both homogeneous and heterogeneous graphs (in Table 1 and Table 2). Specifically, on homogeneous graphs (Table 1), GAE-PS2, GraphSage-PS2, and SEAL-PS2 achieve better results than GAE, GraphSage, and SEAL across two evaluation metrics. Our model has different impacts concerning various backbones. For example, GraphSage-PS2 significantly outperforms GraphSage with up to 9.6% improvements. In heterogeneous scenarios, NGCF-PS2 and LightGCN-PS2 generally perform better than NGCF and LightGCN on three datasets. In particular, the performance gap between our model and two baselines increases on top-10 based metrics. This

Table 1: Link prediction results on Planetoid data. "+"/"-" in the bracket indicates relative improvement with baselines.

	Cora		CiteSeer		PubMed	
	AUC	AP	AUC	AP	AUC	AP
GAE	91.08 ± 0.01	92.03 ± 0.03	89.52 ± 0.04	89.95 ± 0.05	96.40 ± 0.01	96.50 ± 0.02
GAE-RS	89.38 ± 0.23(−1.8%)	91.66 ± 0.45(−0.4%)	89.81 ± 0.29(0.3%)	91.39 ± 0.22(1.6%)	95.34 ± 0.23(−1.1%)	96.13 ± 0.18(−0.3%)
GAE-PS2	92.25 ± 0.71(+1.4%)	93.60 ± 0.25(+1.7%)	92.16 ± 0.19(+3.0%)	93.07 ± 0.06(+3.6%)	98.27 ± 0.10(+2.0%)	98.10 ± 0.16(+1.7%)
GraphSage	86.36 ± 1.06	88.22 ± 0.87	85.24 ± 2.56	86.60 ± 2.54	87.61 ± 0.87	89.41 ± 0.82
GraphSage-RS	89.27 ± 0.70(+3.3%)	89.67 ± 0.58(+1.6%)	88.30 ± 1.08(+3.5%)	89.23 ± 0.91(+3.0%)	88.72 ± 0.77(+1.3%)	90.58 ± 0.75(+1.3%)
GraphSage-PS2	93.86 ± 0.01(8.7%)	93.35 ± 0.25(5.8%)	93.46 ± 1.20(9.6%)	93.62 ± 1.10(8.1%)	93.47 ± 0.22(6.7%)	93.52 ± 0.25(4.6%)
SEAL	90.82 ± 1.97	92.18 ± 0.82	88.49 ± 1.22	90.64 ± 1.46	97.57 ± 0.05	97.20 ± 0.03
SEAL-RS	88.55 ± 0.88(−2.5%)	90.37 ± 0.36(−1.9%)	86.70 ± 1.34(−2.0%)	88.61 ± 1.62(−2.2%)	94.39 ± 0.35(−3.2%)	94.08 ± 0.27(−3.2%)
SEAL-PS2	92.31 ± 0.31(+1.6%)	93.53 ± 0.22(+1.4%)	90.29 ± 0.36(+2.0%)	92.42 ± 0.20(+1.9%)	97.60 ± 0.10(+0.0%)	97.44 ± 0.05(+0.2%)

Table 2: Link prediction results on recommendation datasets.

	Gowalla		Yelp		Amazon-book	
	Hit@10	ndgc@50	Hit@10	ndgc@50	Hit@10	ndgc@50
NGCF	85.16 ± 0.58	64.20 ± 0.34	80.17 ± 0.61	55.12 ± 0.44	70.15 ± 0.59	50.63 ± 0.64
NGCF-RS	83.55 ± 0.47(−1.8%)	62.74 ± 0.40(−2.2%)	78.07 ± 0.53(−2.6%)	52.90 ± 0.32(−4.0%)	67.82 ± 0.65(−3.3%)	48.02 ± 0.81(−5.1%)
NGCF-PS2	88.21 ± 0.24(+3.6%)	65.59 ± 0.60(+1.4%)	87.72 ± 0.35(+9.4%)	56.61 ± 0.27(+2.7%)	78.66 ± 0.30(+12.1%)	51.43 ± 0.41(+1.6%)
LightGCN	86.96 ± 0.46	68.11 ± 0.18	82.44 ± 0.37	58.94 ± 0.28	74.32 ± 0.36	54.79 ± 0.24
LightGCN-RS	85.85 ± 0.39(−1.3%)	66.69 ± 0.20(−2.1%)	81.72 ± 0.57(−0.9%)	57.88 ± 0.35(−1.8%)	73.75 ± 0.38(−0.8%)	53.51 ± 0.18(−2.3%)
LightGCN-PS2	89.79 ± 0.25(+3.3%)	69.75 ± 0.20(+2.4%)	88.44 ± 0.32(+7.3%)	59.99 ± 0.19(+1.8%)	81.26 ± 0.26(+9.3%)	59.29 ± 0.33(+8.2%)

result verifies the effectiveness of our model in accurately recalling related items in the top-ranking list.

Obs. 2. Across various datasets, the proposed personalized subgraph selector consistently outperforms random search-based variants. For different datasets and scenarios (in Table 1 and Table 2), our model consistently outperforms the random search based variants with a large margin. Specifically, random search-based variants are not robust across various datasets. For example, GraphSage-RS could generally achieve better or comparable results with their counterparts on Cora, CiteSeer, and PubMed datasets. But it loses to their counterparts on recommendation datasets in all cases (see Table 2). This comparison validates our motivation to design an automated subgraph selector in a data-driven fashion.

Table 3: Link prediction performance on OGB datasets.

	ogbl-ddi Hit@20	ogbl-collab Hit@50	cogbl-ppa Hit@100
GAE	37.07 ± 5.07	44.75 ± 1.07	18.67 ± 1.32
GAE-RS	30.72 ± 6.86	34.51 ± 0.68	12.65 ± 1.22
GAE-PS2	**49.53 ± 5.99**	**50.26 ± 0.32**	**20.50 ± 0.79**
GraphSage	53.90 ± 4.74	54.63 ± 1.12	16.55 ± 2.40
GraphSage-RS	27.17 ± 5.74	37.54 ± 0.37	9.89 ± 3.46
GraphSage-PS2	**56.90 ± 5.32**	**55.71 ± 0.93**	17.88 ± 1.33
SEAL	30.56 ± 3.86	63.64 ± 0.71	48.80 ± 3.16
SEAL-RS	24.58 ± 4.65	43.56 ± 1.30	35.68 ± 5.21
SEAL-PS2	**32.77 ± 2.50**	**64.83 ± 0.54**	**50.25 ± 2.33**

Obs. 3. The proposed PS2 scales up well on large-scale datasets. On three challenging OGB datasets, our model PS2 can continuously boost the performance of vanilla GNNLP methods, as shown in Table 3. Specifically, GAE-PS2 improves 60.5%, 14.5%,

and 9.8% over GAE on ogbl-ddi, ogbl-collab, and ogbl-ppa datasets, respectively. In contrast, the random search-based variants lose to their counterparts on these three datasets. This observation further demonstrates the effectiveness of considering learnable subgraph selection on large graphs.
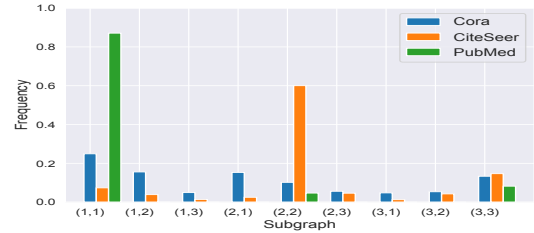
Figure 4: Subgraph distribution of GAE-PS2 on the Plantetoid.

5.3 Subgraph Distribution Analysis

We visualize the learned subgraph distributions of GAE-PS2 on all datasets in Figure 4 and Figure 7 in Appendix to study (**Q2**). By comparing the distributions across different benchmarks, we have the following observation.

Obs. 4. By learning from the data, PS2 can effectively learn different subgraph distributions for various datasets, and even skip some suboptimal subgraphs. Our model PS2 can identify different subgraphs for different edges, and allow different datasets to have their own subgraph distributions (see Figure 4). Specifically, the subgraph distribution on OGB datasets is more sparse than the other two types of datasets, while recommendation datasets generally tend to have smoother distribution. One promising property of PS2 is that it can skip some subgraphs if they are

not optimal for any edges. For example, no edges are assigned to the subgraph (2, 2) on ogbl-ddi and ogbl-ppa datasets.
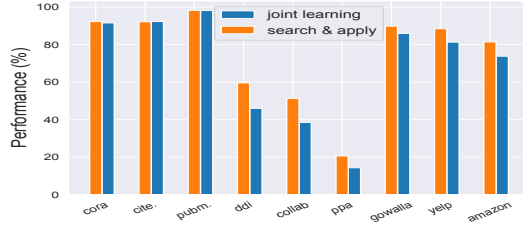

Figure 5: The performance of GAE-PS2 under different optimization paradigms. The evaluation metrics for (Cora, CiteSeer, and Pubmed) and other datasets are AUC and Hit ratio.

5.4 Optimization Analysis

To examine the influence of optimization strategies on PS2 (**Q3**), we compare the default *search & apply* paradigm with the *joint learning* schema on node2link based models. Here, jointly learning means we directly train PS2 with a downstream inference model, i.e., GAE, end-to-end. In this setting, the PS2 training still uses the mixed selection, while the GAE optimization exploits the most likely subgraph via maximum discrete selection. Figure 5 shows the results of two settings on GAE-PS2 over all datasets. We can observe that although *joint learning* strategy achieves comparable results with *search & apply* schema on Cora, CiteSeer, and PubMed datasets, the later schema performs better on the other six large-scale datasets. The possible reason is that joint learning is hard to optimize since the personalized selector and the downstream model are entangled. This comparison validates our choice to adopt the search & apply fashion similar to the AutoML [27] domain.

Besides, we also explore the effectiveness of finetune strategy to avoid the subgraph imbalance issue when applying PS2 for node2link-based methods. Table 6 in Appendix reports the results on Planetoid datasets. Similar observations could be made on other datasets. From Table 6, we observe that finetune strategy outperforms training from scratch on GAE and GraphSage backbones.

5.5 Hyperparameter Analysis

To answer **Q4**, we study the impact of K and hidden dimension D of the score function on PubMed. Figure 6 shows the results on GAE-PS2 and GraphSage-PS2. From the two subfigures, we can observe that our model performs relatively stable over a wide range of combinations of K and D. Specifically, the best results in two cases are achieved when K and D are around 3 and 256, respectively. Similar observations are obtained on other datasets. In experiments, we fix $K = 3$ and set $D = 256$ and $D = 512$ for Planetoid and other datasets (OGB and recommendation), respectively.

5.6 Efficiency Analysis

To study **Q5**, we analyze the training costs of two representatives GNNLPs (GAE and SEAL) after plugging in our personalized selector. For SEAL, we exclude the sampling cost since it is far more than its forward pass running costs. From Table 7 of Appendix, we observe that the additional costs to activate specific subgraph using our personalized selector is marginal, i.e., usually less than
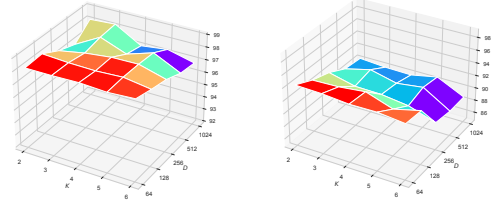

Figure 6: Hyperparameter study of PS2 on BlogCatalog with different base models: GAE (left) and GraphSage (right).

20% running consumption. This is because our selector is simple MLPs, and we can directly generate subgraph embeddings based on GNN output, thanks to the embedding approximation trick in Section 3.2.2.

6 RELATED WORK

In this paper, we mainly focus on graph neural networks (GNNs) based link prediction (GNNLP) techniques. For methods beyond GNN, please refer to [1, 44] for a comprehensive review. For illustration purposes, the existing methods can be mainly divided into two categories: node2link [14, 15, 45] and subgraph2link [18, 19, 21].

node2link is the classical approach to perform link prediction based on GNNs. Given a query edge, it works by first generating representations for two end entities based on their local subgraphs via the GNNs encoder, and then combining the two representations to estimate the edge existence probability. Some efforts have been made to predict missing edges for homogeneous graphs [14, 15, 46, 47], while several methods propose to tackle link prediction on heterogeneous graphs, such as recommendation systems [8, 16, 17, 48, 49] and knowledge graph completion [4].

subgraph2link is a recently proposed new link prediction paradigm. The key idea is to represent each edge with a subgraph around it, and then apply GNNs to learn representation for the whole subgraph. The pioneering work of [18] adopts node labeling to first create structure-aware features for nodes in the subgraph, and then pool over the node representations obtained by GNNs to get the final edge representation. A follow-up work [19] analyzes the impacts of different labeling techniques. Another recent work [21] proposes to replace the pooling operation with a more advanced yet complicated random-walk-based pooling strategy.

7 CONCLUSION

In this paper, we explore a new perspective to train link prediction models by considering edge personalization in terms of neighborhood subgraphs. Specifically, we propose an effective personalized subgraph selector (PS2) as a plug-and-play framework for the graph neural network based link prediction (GNNLP) community. PS2 can automatically and inductively identify optimal subgraph orders for different edges when performing GNNLP. Extensive experiments on multiple datasets with various domains and scales demonstrate the superiority of PS2 against diverse GNNLP backbones. In the future, we will extend PS2 to perform subgraph order selection and critical neighbor sampling within the selected subgraph jointly.

ACKNOWLEDGMENTS

REFERENCES

[1] Tao Zhou. Progresses and challenges in link prediction. *arXiv preprint arXiv:2102.11472*, 2021.

[2] Yanjun Qi, Ziv Bar-Joseph, and Judith Klein-Seetharaman. Evaluation of different biological data and computational classification methods for use in protein interaction prediction. *Proteins: Structure, Function, and Bioinformatics*, 63(3):490–500, 2006.

[3] Zachary Stanfield, Mustafa Coşkun, and Mehmet Koyutürk. Drug response prediction as a link prediction problem. *Scientific reports*, 7(1):1–13, 2017.

[4] Siddhant Arora. A survey on graph neural networks for knowledge graph completion. *arXiv preprint arXiv:2007.12374*, 2020.

[5] Dong Junnan, Qinggang Zhang, Xiao Huang, Qiaoyu Tan, Daochen Zha, and Zihao Zhao. Active ensemble learning for knowledge graph error detection. In *WSDM*.

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

[7] Qiaoyu Tan, Ninghao Liu, and Xia Hu. Deep representation learning for social network analysis. *Frontiers in big Data*, 2:2, 2019.

[8] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. In *KDD*, pages 974–983, 2018.

[9] Qiaoyu Tan, Jianwei Zhang, Ninghao Liu, Xiao Huang, Hongxia Yang, Jingren Zhou, and Xia Hu. Dynamic memory based attention network for sequential recommendation. In *AAAI*, volume 35, pages 4384–4392, 2021.

[10] Daochen Zha, Louis Feng, Qiaoyu Tan, Zirui Liu, Kwei-Herng Lai, Bhargav Bhushanam, Yuandong Tian, Arun Kejariwal, and Xia Hu. Dreamshard: Generalizable embedding table placement for recommender systems. *arXiv preprint arXiv:2210.02023*, 2022.

[11] Chen Gao, Yu Zheng, Nian Li, Yinfeng Li, Yingrong Qin, Jinghua Piao, Yuhan Quan, Jianxin Chang, Depeng Jin, Xiangnan He, et al. Graph neural networks for recommender systems: Challenges, methods, and directions. *arXiv preprint arXiv:2109.12843*, 2021.

[12] Lei Cai, Jundong Li, Jie Wang, and Shuiwang Ji. Line graph neural networks for link prediction. *TPAMI*, 2021.

[13] Renchi Yang, Jieming Shi, Xiaokui Xiao, Yin Yang, and Sourav S Bhowmick. Homogeneous network embedding for massive graphs via reweighted personalized pagerank. *arXiv preprint arXiv:1906.06826*, 2019.

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

[15] William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, pages 1025–1035, 2017.

[16] Xiang Wang, Xiangnan He, Meng Wang, Fuli Feng, and Tat-Seng Chua. Neural graph collaborative filtering. In *SIGIR*, pages 165–174, 2019.

[17] Xiangnan He, Kuan Deng, Xiang Wang, Yan Li, Yongdong Zhang, and Meng Wang. Lightgcn: Simplifying and powering graph convolution network for recommendation. In *SIGIR*, pages 639–648, 2020.

[18] Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. *NIPS*, 31:5165–5175, 2018.

[19] 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. *NIPS*, 34, 2021.

[20] Qiaoyu Tan, Ninghao Liu, Xing Zhao, Hongxia Yang, Jingren Zhou, and Xia Hu. Learning to hash with graph neural networks for recommender systems. In *WWW*, pages 1988–1998, 2020.

[21] Liming Pan, Cheng Shi, and Ivan Dokmanić. Neural link prediction with walk pooling. *arXiv preprint arXiv:2110.04375*, 2021.

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

[23] Xin Zhang, Qiaoyu Tan, Xiao Huang, and Bo Li. Graph contrastive learning with personalized augmentation. *arXiv preprint arXiv:2209.06560*, 2022.

[24] Luca Maria Aiello, Alain Barrat, Ciro Cattuto, Rossano Schifanella, and Giancarlo Ruffo. Link creation and information spreading over social and communication ties in an interest-based online social network. *EPJ Data Science*, 1(1):1–31, 2012.

[25] Ninghao Liu, Qiaoyu Tan, Yuening Li, Hongxia Yang, Jingren Zhou, and Xia Hu. Is a single vector enough? exploring node polysemy for network embedding. In

[26] Qiaoyu Tan, Jianwei Zhang, Jiangchao Yao, Ninghao Liu, Jingren Zhou, Hongxia Yang, and Xia Hu. Sparse-interest network for sequential recommendation. In *WSDM*, pages 598–606, 2021.

[27] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

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

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

[30] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In *AAAI*, volume 33, pages 4780–4789, 2019.

[31] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *CVPR*, pages 8697–8710, 2018.

[32] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. *NIPS*, 25, 2012.

[33] Daochen Zha, Kwei-Herng Lai, Qiaoyu Tan, Sirui Ding, Na Zou, and Xia Ben Hu. Towards automated imbalanced learning with deep hierarchical reinforcement learning. In *CIKM*, pages 2476–2485, 2022.

[34] Jingkang Wang, Tianyun Zhang, Sijia Liu, Pin-Yu Chen, Jiacen Xu, Makan Fardad, and Bo Li. Towards a unified min-max framework for adversarial exploration and robustness. *arXiv preprint arXiv:1906.03563*, 2019.

[35] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pretraining of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[36] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[37] Rushi Longadge and Snehalata Dongre. Class imbalance problem in data mining review. *arXiv preprint arXiv:1305.1707*, 2013.

[38] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

[39] Weihua Hu, Matthias Fey, Hongyu Ren, Maho Nakata, Yuxiao Dong, and Jure Leskovec. Ogb-lsc: A large-scale challenge for machine learning on graphs. *arXiv preprint arXiv:2103.09430*, 2021.

[40] Dawen Liang, Laurent Charlin, James McInerney, and David M Blei. Modeling user exposure in recommendation. In *WWW*, pages 951–961, 2016.

[41] Ruining He and Julian McAuley. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *WWW*, pages 507–517, 2016.

[42] 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. *arXiv preprint arXiv:2005.00687*, 2020.

[43] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *WWW*, pages 173–182, 2017.

[44] Ajay Kumar, Shashank Sheshar Singh, Kuldeep Singh, and Bhaskar Biswas. Link prediction techniques, applications, and performance: A survey. *Physica A: Statistical Mechanics and its Applications*, 553:124289, 2020.

[45] Qiaoyu Tan, Ninghao Liu, Xiao Huang, Soo-Hyun Choi, Li Li, Rui Chen, and Xia Hu. S2gae: Self-supervised graph autoencoders are gen- eralizable learners with graph masking. In *WSDM*.

[46] Shirui Pan, Ruiqi Hu, Guodong Long, Jing Jiang, Lina Yao, and Chengqi Zhang. Adversarially regularized graph autoencoder for graph embedding. *arXiv preprint arXiv:1802.04407*, 2018.

[47] Baole Ai, Zhou Qin, Wenting Shen, and Yong Li. Structure enhanced graph neural networks for link prediction. *arXiv preprint arXiv:2201.05293*, 2022.

[48] Shiwen Wu, Fei Sun, Wentao Zhang, and Bin Cui. Graph neural networks in recommender systems: a survey. *arXiv preprint arXiv:2011.02260*, 2020.

[49] Huachi Zhou, Qiaoyu Tan, Xiao Huang, Kaixiong Zhou, and Xiaoling Wang. Temporal augmented graph neural networks for session-based recommendations. In *SIGIR*, pages 1798–1802, 2021.