Searching to Extrapolate Embedding for Out-of-graph Node Representation Learning

Zhenqian Shen[1], Yan Wen[1], Lanning Wei[2], Wengang Zhang[3],
Yuanhai Luo[3], Chongwu Wu[3], Quanming Yao[1]
[1] Department of Electronic Engineering, Tsinghua University, Beijing, China
[2] Institute of Computing Technology, Chinese Academy of Sciences;
University of Chinese Academy of Sciences, Beijing, China
[3] OPPO Research, Dongguan, China
{szg22, qyaoaa}@tsinghua.edu.cn, owenwen0531@163.com, weilanning18z@ict.ac.cn,
{zhangwengang, null.luo, wuchongwu}@oppo.com

***Abstract*—Out-of-graph node representation learning aims at learning about newly arrived nodes in a given graph. It has wide applications ranging from community detection, recommendation system to malware detection. Although existing methods can be adapted for out-of-graph node representation learning, real-world challenges such as unavailable in-graph structure and data diversity essentially limit the performance of these methods. Here, we formulate the problem as a neural architecture search problem, requiring embedding extrapolation that generates representations for out-of-graph nodes according to their neighbor node embeddings. We propose *searching to extrapolate embedding (S2E)*, a solution based on neural architecture search, to handle the formulated problem. Firstly, we propose an embedding extrapolating framework containing multiple transition modules and an aggregation module to handle fixed in-graph node embedding for embedding extrapolation. To deal with data diversity, we propose searching extrapolating architecture, where we employ objective transformation to handle non-differentiable evaluation metric and make neural architecture search procedure more efficient. In experiments, we show that S2E achieves outstanding performance in real-world datasets. We further conduct experiments on the proposed search space and search algorithm to verify the effectiveness of our design in S2E.**

***Index Terms*—Graph neural network, Neural architecture search, Graph embedding, Out-of-sample learning**

I. Introduction

Graph-structured data, ranging from social networks [1], malware detection networks [2], to citation networks [3] has been widely used in modeling a myriad of real-world applications. Among these applications, one important problem is to learn about the newly arrived nodes (i.e., *out-of-graph nodes*) to these graphs. For examples, we need to learn about newly registered users in social networks and provide them with services such as news feed or new friend recommendation. In malware detection networks, we need to judge whether a newly downloaded APP is a malware online. In this paper, we mainly consider the embedding extrapolation problem for out-of-graph nodes, where we directly deduce out-of-graph node representation from fixed embedding of nodes in the graph (i.e., *in-graph nodes*). This type of problem often arises, when we have limited conditions to utilize in-graph structure information to adjust in-graph node embeddings. For instance,

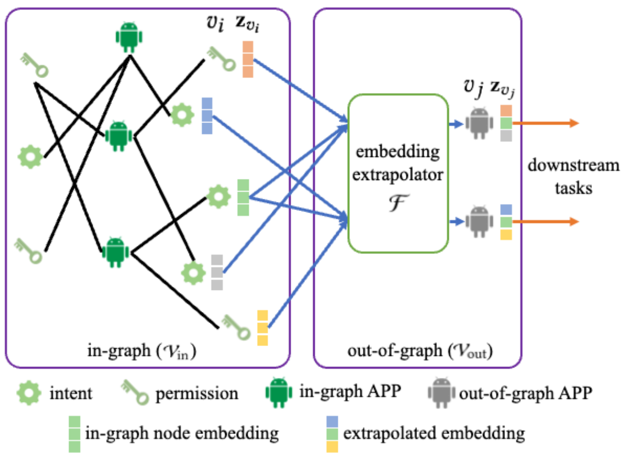

Fig. 1: An example of embedding extrapolation problem for out-of-graph nodes (mobile malware detection).

in applications such as news feed in social networks [4], the output news is needed in a short period of time, where the time consuming in-graph node embedding adaptation through graph structure can not be applied. In applications such as malware detection [2; 5], we have no condition to conduct in-graph node embedding adaptation because graph structure is limited (we do not know which APP contains a certain permission), which means in-graph node embedding must be fixed.

To deal with embedding extrapolation problem for out-of-graph nodes, classic solutions include calculating average among neighbor node embedding or conducting projection based on matrix factorization [6; 7]. However, these methods are usually lack of learning capability, which impede them from achieving good performance. Very recently, Graph Neural Networks (GNNs) have been incorporated for embedding extrapolation problem for out-of-graph nodes, because they can well learn about in-graph node embeddings and make use of links in graph through propagation. Representative GNN methods [8; 9] can be adapted to handle that problem, as they can use aggregation functions to extrapolate in-graph node embeddings. There are also GNN related methods that focus on extrapolation problem for out-of-graph nodes. For example, MEAN [10] proposes a GNN-based propagation model to encode the out-of-knowledge-base entities by in-graph entities' embeddings. Based on MEAN, LAN [11] further introduces logic attention-based aggregation that treats neighbor node embeddings differently when they are propagated to out-of-graph nodes. GEN [12] proposes an embedding extrapolating framework for out-of-graph node representation learning based on few-shot setting.

Although there exists graph embedding extrapolation methods, they are limited by fixed model architectures, making them unable to be adapted to diverse real-world graph-structured data. To address the limitation, we are motivated to propose the following problem: *how can we extrapolate embedding for out-of-graph nodes in a data-dependent way, given fixed in-grade node embeddings?* We propose *searching to extrapolate embedding (S2E)* as our solution: to handle fixed in-graph node embedding, we propose an embedding extrapolating framework that conducts multiple transition steps for the given node embeddings before they are aggregated to the targeted out-of-graph nodes. The design of multiple transition steps can help the model better extrapolates knowledge from the fixed node embeddings; based on embedding extrapolating framework, we design the architecture search space. To tackle data diversity, we propose searching extrapolating architecture. We first transfer the upper level learning objective into a stochastic one to handle non-differentiable evaluation metrics. Moreover, we consider weight sharing via designing supernet and conduct alternative gradient to make architecture search more efficient. S2E can efficiently search for a suitable architecture for our formulated problem, which shows outstanding results compared to existing methods in out-of-graph node classification and link prediction in real-world datasets. We also conduct experiment to understand the designed search space and search algorithm in S2E.

II. Related Works

A. Out-of-graph Node Representation Learning

Graph-structured data holds universal relevance in real-world scenarios, spanning social networks [1], malware detection networks [2], and citation networks [3]. A key challenge associated with graph-structured data is graph node representation learning, which aims to project nodes into a low-dimensional space while preserving their graph properties. In recent years, graph node representation learning has gained significant research interest due to its significant role in various downstream graph learning tasks, such as node classification [13], link prediction [14] and clustering [15]. In the literature, numerous techniques have been proposed for graph node representation learning and representative methods include TransE [16], Deepwalk [17], GCN [18] and Graph-Trans [19].

In this paper, we focus on the problem of embedding extrapolation problem for out-of-graph nodes, which is crucial in the context of dynamic real-world graphs. This problem finds applications in various graph learning tasks, such as community detection in social networks and online malware detection in malware detection networks. Current solution include GNN methods, as they can be extended by designing aggregation functions, like the aggregation function designed in GraphSAGE. Additionally, specific methods have been developed to cater to the requirements of embedding extrapolation problem for out-of-graph nodes. For instance, MEAN [10] and ConvL [20] propose GNN-based propagation models that utilize auxiliary knowledge from in-graph entities to encode out-of-knowledge-base entities, where ConvL further introduces convolution layers to the transition step of the propagation model. Building upon MEAN, LAN [11] introduces a neighborhood aggregator with attention mechanisms to handle diverse relation information between out-of-graph nodes and their in-graph neighbors. GEN [12] proposes an out-of-graph node embedding extrapolation framework based on meta-learning.

When extrapolating embedding for out-of-graph nodes, existing methods mostly use a fixed stacking GNNs, and none of these methods consider data diversity in real-world graphs. In this paper, we take this problem into account and provide a satisfactory solution through embedding extrapolating framework design and architecture search method.

B. Graph Neural Architecture Search

Neural Architecture Search (NAS) is a prominent approach used to find suitable model architectures for diverse datasets within a predefined search space [21]. Early NAS methods employed Reinforcement Learning (RL) methods [22] and Evolutionary Algorithms (EA) [23; 24] to select architectures from the search space, followed by their evaluation in fixed architectures. However, this trial-and-error pipeline is time-consuming, involving the training of thousands of candidate architectures during the search process. Recent advancements have focused on improving efficiency by adopting the weight sharing to re-use the previous trained parameters (a.k.a one-shot NAS methods) [25; 26]. This is achieved through the use of an over-parameterized network (supernet), which encompasses all candidate architectures in the search space. The widely used differentiable NAS methods optimize the parameters by relaxing the discrete space into continuous space and then optimizing the architecture and supernet parameters jointly with gradient descent, and the representative methods are [27; 28; 29].

Motivated by effectiveness NAS, researchers have begun trying to use it to find data-specific graph learning models. Following the message-passing scheme [30], GraphNAS [31] and Auto-GNN [32] take the first step in searching GNNs by adopting the RL-based search algorithm to design the basic components and hyperparameters of common GNNs, e.g., the aggregator, attention function and dimension size. Inspired by recent differentiable NAS methods that improve the search efficiency and effectiveness [27; 28; 29], SANE [33] and DSS [34] propose to search suitable GNNs in a more efficient manner. With the development of GNN methods, there are more graph NAS methods searching for more complicated GNN structures. Apart from stacking aggregation operations, DFG-NAS [35] searches for deep and flexible combination of multiple transition and aggregation layers in GNN. PASCA [36] achieves scalability on large-scale graphs by decoupling the message collection and node representation update process. DiffMG [37] proposes to search for meta-graph to obtain more semantic knowledge from heterogeneous graphs. GNAS-CF [38] proposes to profile the design space of GNN-based Collaborative Filtering in the recommendation system. AutoGEL [39] designs a novel AutoGNN on the link prediction task, which can explicitly model the link information in the graphs. AutoSF [40] aims to automatically design the scoring functions for the knowledge graph embedding learning.

Compared with the aforementioned NAS methods designed under unlimited graph information when updating node representations, our approach proposes a NAS method to handle embedding extrapolation problem for out-of-graph nodes, where neighbor information is limited. We further consider architecture search algorithm based on stochastic relaxation to handle non-differentiable evaluation metric.

III. The Search Problem

A. Notation Definition

Given a graph $\mathcal{G}$, its nodes are divided into in-graph node set $\mathcal{V}_{\text{in}}$ and out-of-graph node set $\mathcal{V}_{\text{out}}$, with $\mathcal{V}_{\text{in}} \cap \mathcal{V}_{\text{out}} = \emptyset$. For $v_i \in \mathcal{V}_{\text{in}}$, we can obtain its embedding $\mathbf{z}_{v_i}$; for $v_j \in \mathcal{V}_{\text{out}}$, we suppose its neighbors in $\mathcal{V}_{\text{in}}$ are known: $N(v_j) = \{v_i | v_i \in \mathcal{V}_{\text{in}}\}_{i=1}^{k_j}$, where k_j is the number of its neighbors. And we denote e_{ij} as the relation of v_j and v_i. As we mainly consider the embedding extrapolation problem, the in-graph node embedding is fixed and we cannot further obtain the neighbor information of nodes in $N(v_j)$. Here we denote neighbor information of v_j as $Q(v_j) = \{(\mathbf{z}_{v_i}, e_{ij}) | v_i \in N(v_j)\}$, which contains in-graph node embeddings and their relations with v_j. Let $\mathbf{x}_{v_j}$ denote the raw features of out-of-graph nodes and we set $\mathbf{x}_{v_j}$ as $\mathbf{0}$ when they are not provided.

To achieve data-dependent embedding extrapolation, here we hope to design an embedding extrapolator with adaptive architecture as follows:

$$\mathbf{z}_{v_j} = \mathcal{F}_{\boldsymbol{\alpha}}(Q(v_j), \mathbf{x}_{v_j}; \boldsymbol{w}),$$

where $\boldsymbol{\alpha}$ and $\boldsymbol{w}$ are the architecture and model parameters of the embedding extrapolator, respectively.

Take the malware detection network in Figure 1 as an example. $\mathcal{V}_{\text{in}}$ includes common intents, permissions and the APPs we already know. For the newly arrived APP, i.e., $v_j \in \mathcal{V}_{\text{out}}$, its neighbors $N(v_j)$ represent its common permissions and intents. The neighbor information $Q(v_j)$ contains the neighbors' attribute vectors and relations on the edges connected to v_j. To extrapolate embedding of neighbor permissions and intents, an effective embedding extrapolator, i.e., $\mathcal{F}_{\boldsymbol{\alpha}}$, is strongly needed.

B. Problem Formulation

Here, we split the $\mathcal{V}_{\text{out}}$ into training, validation and test sets as $\mathcal{V}_{\text{tra}}$, $\mathcal{V}_{\text{val}}$, $\mathcal{V}_{\text{tst}}$. We define $\mathcal{L}$ and $\mathcal{M}$ as the loss function and evaluation metric for the downstream task of learned node representation. In node classifications, cross entropy loss is usually taken as $\mathcal{L}$, accuracy and Macro-F1 are usually taken as $\mathcal{M}$ (details are in Section V-A) [8; 18]. In link prediction tasks, we usually use margin ranking loss as $\mathcal{L}$, Mean Reciprocal Rank (MRR) and top at K (Hit@K) as $\mathcal{M}$ (details are in Section V-B) [41; 42]. Denote $\mathcal{A}$ as the candidate architecture set, $\boldsymbol{w}$ as the general model parameter. The objective is to search an architecture $\boldsymbol{\alpha} \in \mathcal{A}$ such that the embedding extrapolator $\mathcal{F}_{\boldsymbol{\alpha}}$ can achieve the best predicting performance on the validation set $\mathcal{V}_{\text{val}}$.

Definition 1: (S2E Problem) The problem of architecture search to extrapolate embedding for out-of-graph nodes is formulated as

$$\bar{\boldsymbol{\alpha}} = \arg\max_{\boldsymbol{\alpha} \in \mathcal{A}} \sum_{v_j \in \mathcal{V}_{\text{val}}} \mathcal{M}(\mathcal{F}_{\boldsymbol{\alpha}}(Q(v_j), \mathbf{x}_{v_j}; \bar{\boldsymbol{w}})), \tag{1}$$

$$\text{s.t. } \bar{\boldsymbol{w}} = \arg\min_{\boldsymbol{w}} \sum_{v_j \in \mathcal{V}_{\text{tra}}} \mathcal{L}(\mathcal{F}_{\boldsymbol{\alpha}}(Q(v_j), \mathbf{x}_{v_j}; \boldsymbol{w})). \tag{2}$$

To address the aforementioned problem, there are two key points that need to be considered. Firstly, the in-graph node embeddings in $Q(v_j)$ remain fixed, which implies that most existing graph NAS methods [31; 32; 33] cannot be directly applied as they require in-graph node embedding adaptation. Secondly, the evaluation metric $\mathcal{M}$ may be non-differentiable (e.g. MRR and Hit@K in link predictions), which means that gradient-based NAS methods [27; 33; 43] cannot be directly employed in the proposed problem.

IV. The proposed method

In this section, we introduce our solution, searching to extrapolate embedding (S2E), for the problem introduced in Section III. We first design an embedding extrapolating framework to handle the fixed in-graph node embedding. Then, we propose searching extrapolating architecture to deal with data diversity. Within this context, we employ object transformation, which handles the challenge of non-differentiable evaluation metrics and accelerates the architecture search.

A. An Embedding Extrapolating Framework

To handle node embedding extrapolation problem for out-of-graph nodes, we propose an embedding extrapolating framework for each out-of-graph node $v_j \in \mathcal{V}_{\text{out}}$. To fully make use of the fixed in-graph node embedding, we propose transition from in-graph nodes and aggregation module to extrapolate knowledge. For out-of-graph raw features that are possibly given, we introduce the module of transition for out-of-graph nodes. The upper part of Figure 2 displays the architecture of our designed embedding extrapolating framework. In the following part, we denote σ as the activation function and τ as the τth transition layer. Each type of the modules is described in detail as follows:

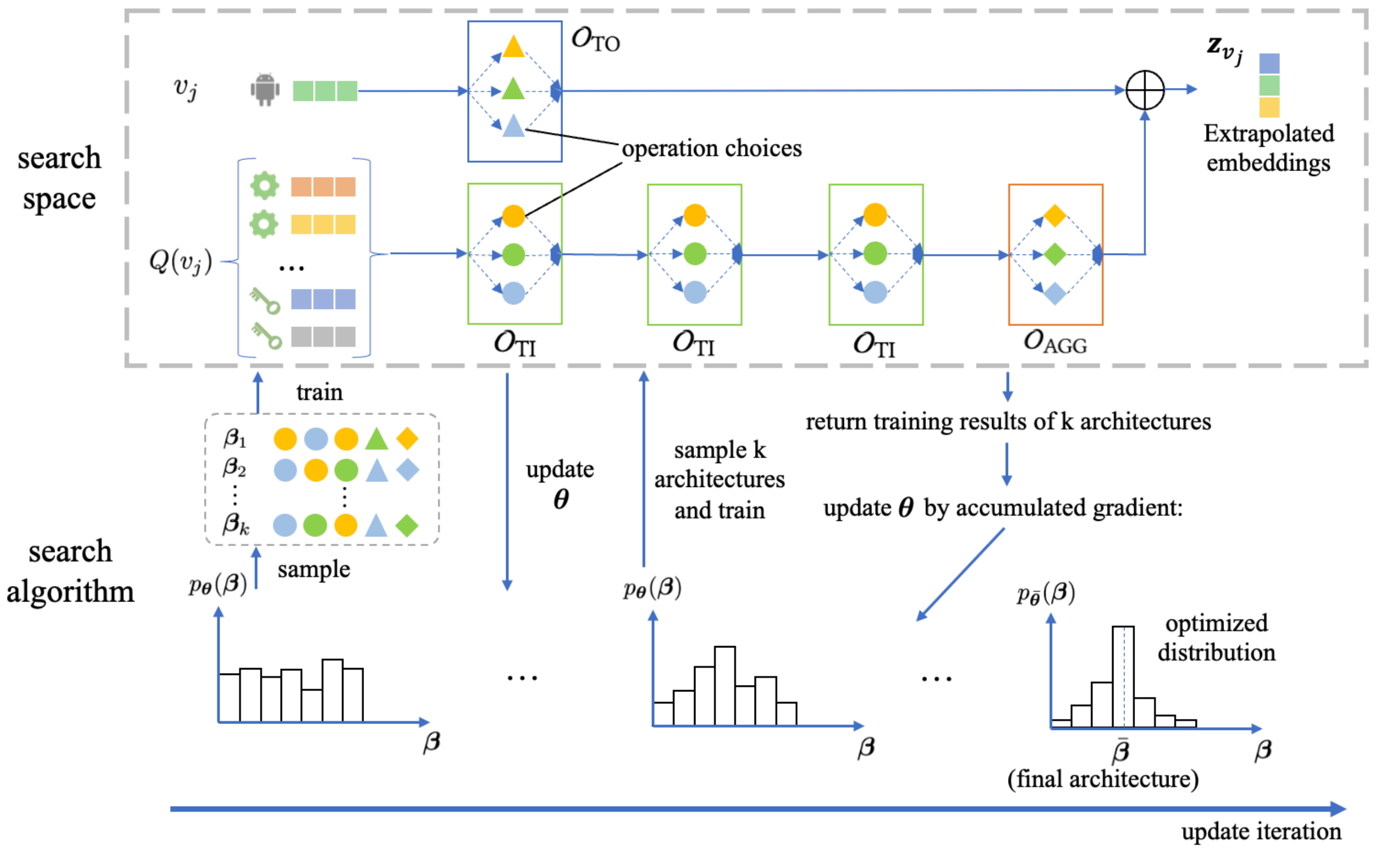

Fig. 2: A figure to illustrate our proposed S2E method. The upper part in the enclosed in the grey dashed box represents the embedding extrapolating framework. $\mathcal{O}_{\text{TI}}$, $\mathcal{O}_{\text{AGG}}$, and $\mathcal{O}_{\text{TO}}$ refer to the transitions from in-graph nodes, aggregation, and transitions from out-of-graph nodes, respectively, with $\oplus$ denoting summation. For the layer number of transition modules, we set $L_{\text{TI}} = 3$ and $L_{\text{TO}} = 1$ in the figure. The operation choices in each module form the search space $\mathcal{B}$. The lower part shows searching extrapolating architecture. Here we introduce distribution parameter $\boldsymbol{\theta}$, and then searching architectures by optimizing $\boldsymbol{\theta}$ instead of directly optimizing the architecture $\boldsymbol{\beta}$. In each update step, we first sample k architectures from the supernet and train them respectively. Then we return the training results of k architectures and update $\boldsymbol{\theta}$ using the accumulated gradient. After obtaining the optimized $\bar{\boldsymbol{\theta}}$, we determine the final architecture $\bar{\boldsymbol{\beta}}$ that has the highest probability in the optimized probability distribution $p_{\bar{\boldsymbol{\theta}}}(\boldsymbol{\beta})$.

- **Transition from in-graph nodes.** This module is used to conduct transition step for in-graph node embedding as follows:

$$\mathbf{z}_{v_i}^{\tau} = \sigma\Big(\mathcal{O}_{\text{TI}}\big(\mathbf{z}_{v_i}^{\tau-1}, e_{ij}; w_{\text{TI}}^{\tau}\big)\Big), \tag{3}$$

  Here we initialize $\mathbf{z}_{v_i}^0 = \mathbf{z}_{v_i}$ and set the layer number of $\mathcal{O}_{\text{TI}}$ as L_{TI}. w_{TI}^{τ} is the parameter of the τth transition layer and $\boldsymbol{W}_{\text{TI}} = \{w_{\text{TI}}^{\tau}\}_{\tau=1}^{L_{\text{TI}}}$ is the set of parameters in $\mathcal{O}_{\text{TI}}$. As for the choice of transition layers, we first consider four basic operations: `IDENTITY`, `Conv1D`, `LINEAR`, `GATING`. `IDENTITY` represents direct connection, `Conv1D` denotes convolution on a single dimension. `LINEAR` represents linear layer, and `GATING` outputs $(\text{ReLU}(\mathbf{W}_1 x) \cdot x) \cdot \mathbf{W}_2$ for the input x. Considering that the relation between in-graph nodes and out-of-graph nodes can be different, we utilize the design of multi-kernel, i.e., using different groups of parameters for different kinds of relations. We add the multi-kernel design for the operation `Conv1D` and `LINEAR`, which we denote as `Conv1D-r` and `LINEAR-r`.
- **Aggregation.** The aggregation module mainly conducts the aggregation step from the in-graph nodes to out-of-graph nodes as follows:

$$\mathbf{z}_{v_j}^{L_{\text{TI}}} = \mathcal{O}_{\text{AGG}}\Big(\mathbf{z}_{v_1}^{L_{\text{TI}}}, \mathbf{z}_{v_2}^{L_{\text{TI}}}, ..., \mathbf{z}_{v_{k_j}}^{L_{\text{TI}}}; w_{\text{AGG}}\Big), \tag{4}$$

  where $\boldsymbol{W}_{\text{AGG}} = w_{\text{AGG}}$ denotes the parameters of aggregation. Here we mainly consider the aggregation operation, including `SAGE-SUM`, `SAGE-MEAN`, `SAGE-MAX` [8] and `GAT` [9].
- **Transition for out-of-graph nodes.** This module is used for transition step for out-of-graph node raw features when they are provided:

$$\mathbf{x}_{v_j}^{\tau} = \sigma\Big(\mathcal{O}_{\text{TO}}\big(\mathbf{x}_{v_j}^{\tau-1}; w_{\text{TO}}^{\tau}\big)\Big), \tag{5}$$

  Here, we initialize $\mathbf{x}_{v_j}^0 = \mathbf{x}_{v_j}$ and set the layer number of $\mathcal{O}_{\text{TO}}$ as L_{TO}. w_{TO}^{τ} is the parameter of the τth transition layer and $\boldsymbol{w}_{\text{TO}} = \{w_{\text{TO}}^{\tau}\}_{\tau=1}^{L_{\text{TO}}}$ is the set of parameters in $\mathcal{O}_{\text{TO}}$. As the transition operation here is irrelevant with relations between nodes, here we choose the four basic transition operations: `IDENTITY`, `Conv1D`, `LINEAR`, `GATING`.

Combining the parameters in three types of modules, the model parameter of the embedding extrapolator is $\boldsymbol{W} = \{\boldsymbol{W}_{\text{TI}}, \boldsymbol{W}_{\text{AGG}}, \boldsymbol{W}_{\text{TO}}\}$. When out-of-graph node raw features are provided, the output of our embedding extrapolator is $\mathbf{z}_{v_j} = \mathbf{z}_{v_j}^{L_{\text{TI}}} + \mathbf{x}_{v_j}^{L_{\text{TO}}}$, otherwise the output is $\mathbf{z}_{v_j} = \mathbf{z}_{v_j}^{L_{\text{TI}}}$, which is regarded as the output of the embedding extrapolator.

The modules mentioned above with operation choices in Eq.(3)-(5) form proposed search space $\mathcal{B}$ based on the embedding extrapolating framework, and the choices of each module are summarized in Table I. When considering the search space with $L_{\text{TI}} = 3$ and $L_{\text{TO}} = 1$, the total number of candidate architectures is $6^3 \times 4 \times 4 = 3,456$, which is a moderate search space size.

TABLE I: Choices for transition and aggregation operation

Module	Operations
$\mathcal{O}_{\textbf{TI}}$	`IDENTITY`, `Conv1D`, `LINEAR`, `GATING`, `Conv1D-r`, `LINEAR-r`
$\mathcal{O}_{\textbf{TO}}$	`IDENTITY`, `Conv1D`, `LINEAR`, `GATING`
$\mathcal{O}_{\textbf{AGG}}$	`SAGE-SUM`, `SAGE-MEAN`, `SAGE-MAX`, `GAT`

B. Searching Extrapolating Architecture

To realize a data-dependent embedding extrapolator, it becomes essential to propose an approach for effective architecture search. Here, we consider the strategy of gradient descent [27; 28] for the optimization of the architecture, as it is more efficient than traditional architecture search methods such as random search and Bayesian optimization. To deal with non-differentiable evaluation metric in upper level objective, we transform the initial upper level objective into a stochastic one through re-parameterization. Furthermore, we introduce supernet that shares weight among possible architectures to make the architecture search procedure more efficient.

1) Objective Transformation: Given that the evaluation metric $\mathcal{M}$ in upper-level objective of Definition 1 is probably non-differentiable, we transform $\mathcal{M}$ into a stochastic objective $\mathcal{J}$ via idea of re-parameterization [25; 44]. To achieve that, we introduce probability distribution $p_{\boldsymbol{\theta}}(\boldsymbol{\beta})$ parameterized by $\boldsymbol{\theta}$ for each architecture $\boldsymbol{\beta} \in \mathcal{B}$. Let

$$\tilde{\mathcal{M}}(\bar{\boldsymbol{w}}, \boldsymbol{\beta}) = \sum_{v_j \in \mathcal{V}_{\text{val}}} \mathcal{M}(\mathcal{F}_{\boldsymbol{\beta}}(Q(v_j), \mathbf{x}_{v_j}; \bar{\boldsymbol{w}})),$$

where $\bar{\boldsymbol{w}}$ is the optimized model parameter. The stochastic relaxation of $\mathcal{M}$ given $p_{\boldsymbol{\theta}}(\boldsymbol{\beta})$ is as follows:

$$\mathcal{J}(\boldsymbol{\theta}) = \sum_{\boldsymbol{\beta} \in \mathcal{B}} \tilde{\mathcal{M}}(\bar{\boldsymbol{w}}, \boldsymbol{\beta}) p_{\boldsymbol{\theta}}(\boldsymbol{\beta}) = \mathbb{E}_{p_{\boldsymbol{\theta}}}\left[\tilde{\mathcal{M}}(\bar{\boldsymbol{w}}, \boldsymbol{\beta})\right].$$

The following lemma shows that maximizing $\mathcal{J}$ is equivalent to maximizing the initial upper level objective $\mathcal{M}$ in (1):

Lemma 4.1: Assume that for any $\bar{\boldsymbol{\beta}}$, there exists a $\boldsymbol{\theta}$ (could be infinite) such that $p_{\boldsymbol{\theta}}(\bar{\boldsymbol{\beta}}) = 1$. Then, (i) $\sup_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}) = \max_{\boldsymbol{\beta}} \tilde{\mathcal{M}}(\bar{\boldsymbol{w}}, \boldsymbol{\beta})$; (ii) $\bar{\boldsymbol{\theta}}$ that maximizes $\mathcal{J}(\boldsymbol{\theta})$ satisfies $p_{\bar{\boldsymbol{\theta}}}(\arg\max_{\boldsymbol{\beta}} \tilde{\mathcal{M}}(\bar{\boldsymbol{w}}, \boldsymbol{\beta})) = 1$.

Through stochastic relaxation, the problem of non-differentiable evaluation metric can be handled, because the gradient of $\mathcal{J}(\boldsymbol{\theta})$ can be obtained as:

$$\nabla_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}) = \mathbb{E}_{p_{\boldsymbol{\theta}}}\Big[\tilde{\mathcal{M}}(\bar{\boldsymbol{w}}, \boldsymbol{\beta}) \nabla \log(p_{\boldsymbol{\theta}}(\boldsymbol{\beta}))\Big].$$

Thus, we can optimize the new objective $\mathcal{J}(\boldsymbol{\theta})$ instead of directly optimizing $\mathcal{M}$. Building upon initial upper level objective in (1), we can transfer it as the following objective:

$$\bar{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}). \tag{6}$$

To optimize the transformed objective (6), we need to instantiate the probability $p_{\boldsymbol{\theta}}(\boldsymbol{\beta})$. Given all choices for each module are discrete in Table I, we use softmax-liked distribution to represent the probability to sample a specific operation for each module. For example, consider L_{TO} $\mathcal{O}_{\text{TO}}$ modules with their operation choice set $\mathcal{C}_{\text{TO}}$, Here, we denote the choice of L_{TO} modules as $\boldsymbol{\beta}_{\text{TO}} = \{\beta_{\text{TO}}^{\tau}\}_{\tau=1}^{L_{\text{TO}}}$, where $\beta_{\text{TO}}^{\tau} \in \mathcal{C}_{\text{TO}}$. The probability to choose $\boldsymbol{\beta}_{\text{TO}}$ is $p_{\text{TO}}(\boldsymbol{\beta}_{\text{TO}}) = \prod_{\tau=1}^{L_{\text{TO}}} \exp(\theta_{\text{TO}}^{\tau,o}) / \sum_{o' \in \mathcal{C}_{\text{TO}}} \exp(\theta_{\text{TO}}^{\tau,o'})$, where $o = \beta_{\text{TO}}^{\tau}$ and $\theta_{\text{TO}}^{\tau,o}$ is the learnable probability distribution parameter. For $\mathcal{O}_{\text{TO}}$, we form probability distribution parameter set as $\boldsymbol{\theta}_{\text{TO}} = \{\theta_{\text{TO}}^{\tau,o} | o \in \mathcal{C}_{\text{TO}}\}_{\tau=1}^{L_{\text{TO}}}$. $\boldsymbol{\beta}_{\text{TI}}$, $\boldsymbol{\beta}_{\text{AGG}}$, $\boldsymbol{\theta}_{\text{TI}}$ and $\boldsymbol{\theta}_{\text{AGG}}$ can be obined in the same way. Combining architectures and probability distribution parameters of three types of modules, the architecture $\boldsymbol{\beta}$ can be denoted as $\boldsymbol{\beta} = \{\boldsymbol{\beta}_{\text{TI}}, \boldsymbol{\beta}_{\text{AGG}}, \boldsymbol{\beta}_{\text{TO}}\} = \{\{\beta_{\text{TI}}^{\tau}\}_{\tau=1}^{L_{TI}}, \beta_{\text{AGG}}, \{\beta_{\text{TO}}^{\tau}\}_{\tau=1}^{L_{TO}}\}$ and distribution parameters $\boldsymbol{\theta} = \{\boldsymbol{\theta}_{\text{TI}}, \boldsymbol{\theta}_{\text{AGG}}, \boldsymbol{\theta}_{\text{TO}}\}$. And the final expression of distribution parameters $p_{\boldsymbol{\theta}}(\boldsymbol{\beta})$ is in the form of successive multiplication:

$$p_{\boldsymbol{\theta}}(\boldsymbol{\beta}) = p_{\text{TO}}(\boldsymbol{\beta}_{\text{TO}}) \cdot p_{\text{TI}}(\boldsymbol{\beta}_{\text{TI}}) \cdot p_{\text{AGG}}(\boldsymbol{\beta}_{\text{AGG}}).$$

Then, we design architecture optimization algorithm. Directly optimizing the lower-level objective in (2) requires training the model parameters of each sampled architecture from scratch, which is a highly costly and time-consuming process. To handle that problem, we consider weight sharing [25; 27]. We introduce a supernet with embedding extrapolator parameter $\boldsymbol{W} = \{\boldsymbol{W}_{\text{TI}}, \boldsymbol{W}_{\text{AGG}}, \boldsymbol{W}_{\text{TO}}\} = \{\{w_{\text{TO}}^{\tau}\}_{\tau=1}^{L_{\text{TO}}}, \{w_{\text{AGG}}\}, \{w_{\text{TI}}^{\tau}\}_{\tau=1}^{L_{\text{TI}}}\}$, which contains all the possible choices in each module. For each possible architecture $\boldsymbol{\beta}$, its corresponding model parameter in the supernet is denoted as $\boldsymbol{W}(\boldsymbol{\beta})$. To achieve efficient architecture search, we collectively optimize parameter $\boldsymbol{W}$ of the supernet, instead of training each sampled network from scratch. This strategy significantly reduces the computational burden and accelerates the search process. After optimizing $\boldsymbol{W}$ as $\bar{\boldsymbol{W}}$, we use $\bar{\boldsymbol{W}}(\boldsymbol{\beta})$ to approximate the optimal model parameter of each single architecture.

Combining the upper-level objective transformation and lower-level objective approximation, the search problem in Definition 1 is transformed as follows:

$$\begin{aligned}\bar{\boldsymbol{\theta}} &= \arg\max_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}),\\ \text{s.t. } \bar{\boldsymbol{W}}(\boldsymbol{\beta}) &= \arg\min_{\boldsymbol{W}(\boldsymbol{\beta})} \sum_{v_j \in \mathcal{V}_{\text{tra}}} \mathcal{L}(\mathcal{F}_{\boldsymbol{\beta}}(Q(v_j), \mathbf{x}_{v_j}; \boldsymbol{W}(\boldsymbol{\beta}))),\end{aligned}$$

where $\mathcal{J}(\boldsymbol{\theta}) = \mathbb{E}_{p_{\boldsymbol{\theta}}}\big[\tilde{\mathcal{M}}(\bar{\boldsymbol{W}}(\boldsymbol{\beta}), \boldsymbol{\beta})\big]$.

2) Alternative gradient descent: To handle the search problem, we consider to optimize $\boldsymbol{W}$ and $\boldsymbol{\theta}$ iteratively. In contrast to existing NAS methods [27], our approach has two process. First, we sample architectures based on the probability distribution $p_{\boldsymbol{\theta}}(\boldsymbol{\beta})$. Then, we update the parameters by accumulating the gradients collected from each sampled architecture.

In each update step, we first sample k architectures from the distribution $p_{\boldsymbol{\theta}}(\boldsymbol{\beta})$, which we denote as $\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, ..., \boldsymbol{\beta}_k$. Here we first introduce our strategy to update $\boldsymbol{W}$. For brevity, we take the update of the τth $\mathcal{O}_{\text{TO}}$ module as an example. Firstly, we calculate the gradient of the loss function with regard to the module as

$$\boldsymbol{h}_i = \nabla_w \sum_{v_j \in \mathcal{V}_{\text{tra}}} \mathcal{L}(\mathcal{F}_{\boldsymbol{\beta}_i}(Q(v_j), \mathbf{x}_{v_j}; \boldsymbol{W}(\boldsymbol{\beta}_i))), \tag{7}$$

where $\beta_i = (\boldsymbol{\beta}_i)^{\tau}_{\text{TO}}$ and $w = w^{\tau}_{\text{TO}}$. Then we conduct the update step of w. Let us denote $g(i) = |\{\mathbb{1}(\beta_j, \beta_i) | 1 \leq j \leq k\}|$, which represents the number of times operation β_i has been sampled in this module. The update scheme of w is as follows:

$$w = w + \xi_{\boldsymbol{W}} \sum_{i=1}^{k} \frac{1}{g(i)} \boldsymbol{h}_i, \tag{8}$$

where $\xi_{\boldsymbol{W}}$ is the learning rate for $\boldsymbol{W}$. The complete update procedure of $\boldsymbol{W}$ can be refer to Appendix.

For the distribution parameter $\boldsymbol{\theta}$, we update it as follows:

$$\boldsymbol{\theta} = \boldsymbol{\theta} + \xi_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}), \tag{9}$$

where $\xi_{\boldsymbol{\theta}}$ is the learning rate for $\boldsymbol{\theta}$ and we approximate the gradient $\nabla_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta})$ as follows:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}) &= \mathbb{E}_{p_{\boldsymbol{\theta}}}\Big[\tilde{\mathcal{M}}(\boldsymbol{W}(\boldsymbol{\beta}), \boldsymbol{\beta}) \nabla \log(p_{\boldsymbol{\theta}}(\boldsymbol{\beta}))\Big],\\ &\approx \frac{1}{k} \sum_{i=1}^{k} \tilde{\mathcal{M}}(\boldsymbol{W}(\boldsymbol{\beta}_i), \boldsymbol{\beta}_i) \nabla_{\boldsymbol{\theta}} \log(p_{\boldsymbol{\theta}}(\boldsymbol{\beta}_i)).\end{aligned}$$

After obtaining the optimal $\bar{\boldsymbol{\theta}}$, we can choose the $\bar{\boldsymbol{\beta}}$ with the largest probability as the searched model architecture. Finally, we retrain the searched architecture $\bar{\boldsymbol{\beta}}$ from scratch and then report the model performance.

C. Complete Algorithm

The complete procedure of proposed S2E method is summarized in Algorithm 1. In each update iteration, Step 3-4 sample k architectures and build corresponding embedding extrapolators from the supernet. After training each embedding extrapolator, Step 5-6 update $\boldsymbol{W}$ and $\boldsymbol{\theta}$ according to accumulated gradient. With optimized $\bar{\boldsymbol{\theta}}$, $\bar{\boldsymbol{\beta}}$ with the largest probability in $p_{\bar{\boldsymbol{\theta}}}(\boldsymbol{\beta})$ is selected as the searched architecture in step 8.

Algorithm 1 Search to Extrapolate Embedding for Out-of-graph Node Representation Learning (S2E)

```
Require: The raw features and neighborhood information of out-of-graph nodes $\{\mathbf{x}_{v_j}, Q(v_j) | \forall v_j \in \mathcal{V}_{\text{out}}\}$. The transition layer numbers $L_{\text{TO}}$ and $L_{\text{TI}}$.
1: Initialize $\boldsymbol{\theta}$ and $\boldsymbol{W}$.
2: while not done do
3:     Draw $k$ architecture parameter $\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, ..., \boldsymbol{\beta}_k$ according to distribution $p_{\boldsymbol{\theta}}(\boldsymbol{\beta})$;
4:     Construct the corresponding embedding extrapolators $\mathcal{F}_{\boldsymbol{\beta}_1}, \mathcal{F}_{\boldsymbol{\beta}_2}, ..., \mathcal{F}_{\boldsymbol{\beta}_k}$ using (3)- (5);
5:     Update $\boldsymbol{W}$ like (7) (8);
6:     Update $\boldsymbol{\theta}$ by (9);
7: end while
8: Choose the $\bar{\boldsymbol{\beta}}$ with the largest probability in $p_{\bar{\boldsymbol{\theta}}}(\boldsymbol{\beta})$ as the output model architecture;
9: return The searched architecture $\bar{\boldsymbol{\beta}}$.
```

Compared to existing methods on embedding extrapolation problem for out-of-graph nodes based on GNN [10; 12; 20], we propose an embedding extrapolating framework, which is flexible for different real world graph learning problems and suitable for the embedding extrapolating problem we focus on. As for architecture search algorithm, we consider gradient descent because of its efficiency. Different from the methods that directly compute gradient for architecture parameters [27; 33; 43], we employ stochastic relaxation and introduce probability parameter for different architectures to handle the problem of non-differentiable evaluation metric, which commonly appears in graph learning downstream tasks. Finally, we optimize the supernet with shared weights across all potential architectures to find the optimal embedding extrapolating architecture.

V. Experiments

In this section, we conduct experiments on both out-of-graph node classification and link prediction in real-world datasets to evaluate the performance of the proposed S2E method. We further conduct experiments to verify the reliability of the search algorithm and the rationality of the proposed search space. Experiments are conducted on 24 GB NVIDIA GeForce RTX 4090 GPU and AMD EPYC 7763 CPU.

A. Node Classification

1) Experiment setting:

Baselines: As the analysis in Section II-A, we mainly choose GNN methods and out-of-graph node representation learning methods as out-of-graph representation learning baselines in our experiment. We choose GAT [9] and GraphSAGE [8] as GNN baseline method, for which we implement a single GNN layer for out-of-graph node representation learning. As for GNN methods target out-of-graph learning,

TABLE II: Comparisons of different methods on out-of-graph node classification task (without raw features). The classification performances with 95% confidence interval are reported. In the table, **I-encoder** represents the choice of in-graph node encoder. The best results are in boldface, and the second best are underlined.

Dataset	**I-encoder**	**Metric**	GAT	GraphSAGE	MEAN	ConvL	GEN	**S2E**
ACM	RGCN	**Macro-F1**	0.543 (0.010)	0.475 (0.013)	0.709 (0.008)	0.672 (0.007)	<u>0.712</u> (0.008)	**0.738 (0.005)**
		Accuracy	0.542 (0.013)	0.471 (0.018)	0.702 (0.008)	0.665 (0.007)	<u>0.707</u> (0.006)	**0.739 (0.006)**
	GAT	**Macro-F1**	0.718 (0.015)	0.632 (0.014)	<u>0.748</u> (0.006)	0.740 (0.006)	0.738 (0.002)	**0.758 (0.004)**
		Accuracy	0.712 (0.017)	0.625 (0.013)	0.733 (0.007)	<u>0.735</u> (0.006)	0.730 (0.004)	**0.752 (0.004)**
	TransE	**Macro-F1**	0.606 (0.017)	0.604 (0.013)	<u>0.700</u> (0.008)	0.669 (0.007)	0.697 (0.010)	**0.718 (0.004)**
		Accuracy	0.607 (0.017)	0.601 (0.018)	<u>0.691</u> (0.005)	0.660 (0.007)	0.691 (0.007)	**0.703 (0.002)**
IMDB	RGCN	**Macro-F1**	0.347 (0.015)	0.317 (0.020)	0.472 (0.010)	0.447 (0.008)	<u>0.480</u> (0.014)	**0.493 (0.005)**
		Accuracy	0.383 (0.017)	0.354 (0.017)	<u>0.521</u> (0.010)	0.471 (0.009)	0.507 (0.012)	**0.536 (0.004)**
	GAT	**Macro-F1**	0.305 (0.017)	0.334 (0.016)	0.380 (0.008)	0.360 (0.006)	<u>0.391</u> (0.006)	**0.413 (0.005)**
		Accuracy	0.371 (0.015)	0.375 (0.017)	<u>0.441</u> (0.009)	0.427 (0.008)	0.432 (0.004)	**0.486 (0.004)**
	TransE	**Macro-F1**	0.360 (0.011)	0.374 (0.014)	<u>0.387</u> (0.007)	0.379 (0.007)	0.380 (0.014)	**0.425 (0.004)**
		Accuracy	0.408 (0.012)	0.404 (0.013)	<u>0.434</u> (0.006)	0.434 (0.006)	0.430 (0.010)	**0.482 (0.006)**
MobAPP	RGCN	**Macro-F1**	0.851 (0.018)	0.938 (0.016)	0.967 (0.007)	<u>0.972</u> (0.008)	0.970 (0.004)	**0.972 (0.005)**
		Accuracy	0.854 (0.017)	0.938 (0.015)	0.967 (0.008)	<u>0.971</u> (0.008)	0.971 (0.003)	**0.972 (0.005)**
	GAT	**Macro-F1**	0.503 (0.017)	0.742 (0.018)	0.882 (0.006)	0.847 (0.007)	<u>0.912</u> (0.008)	**0.922 (0.005)**
		Accuracy	0.518 (0.017)	0.745 (0.016)	0.891 (0.009)	0.851 (0.007)	<u>0.912</u> (0.008)	**0.921 (0.006)**
	TransE	**Macro-F1**	0.656 (0.012)	0.835 (0.014)	0.900 (0.011)	0.907 (0.007)	<u>0.914</u> (0.006)	**0.948 (0.006)**
		Accuracy	0.661 (0.011)	0.836 (0.014)	0.902 (0.008)	0.908 (0.007)	<u>0.915</u> (0.007)	**0.948 (0.005)**

we implement MEAN [10], ConvL [20] and GEN [12]. For the setting with node raw features provided, we further compare with MLP, which merely predict node classes based on node raw features.

Datasets: We perform the experiments of out-of-graph node classification on three datasets (Table III). Detailed descriptions of each dataset are as follows: (1) ACM is a citation graph that contains papers, authors and subjects as nodes. The classification task is to classify the types of papers from *Database, Wireless Communication and Data Mining*. The raw features of authors correspond to the elements of a bag-of-words represented of keywords. (2) IMDB is a graph that records the relations of movies, actors and directors. The task is to classify the types of movies from *Action, Comedy and Drama*. The raw features of movies in the graph are extracted from the elements of a bag-of-words represented of keywords. (3) MobAPP is a mobile application knowledge graph that contains the APPs, intents and permissions as nodes. The classification task is to judge whether an APP is mobile malware.

Given that the out-of-graph node raw features are optional in real-world graphs, we conduct experiments in two settings (without raw features and using raw features).

Dataset processing and inplementation: To satisfy the setting of embedding extrapolation for out-of-graph node introduced in Section III, we further divide each graph data into

TABLE III: Statistics of node classification datasets. For out-of-graph nodes, we present the number of nodes in training/-validation/test set. "-" represent nodes have no raw features in the dataset.

Dataset	# In-graph Nodes	# Out-of-graph Nodes	# Classes	# Raw features
ACM	7,469	300/300/925	3	1,902
IMDB	11,311	300/300/861	3	1,256
MobAPP	30,051	1,400/1,400/4,227	2	-

a in-graph node set ($\mathcal{V}_{\text{in}}$) and a out-of-graph node set ($\mathcal{V}_{\text{out}}$). The out-of-graph node set is further divided into train, valid and test node sets ($\mathcal{V}_{\text{tra}}$, $\mathcal{V}_{\text{val}}$ and $\mathcal{V}_{\text{tst}}$).

In our experiments, in-graph encoders are mainly used to obtain the embeddings of in-graph nodes. We use different kind of in-graph encoders to simulate different given embeddings in real world scenarios. For out-of-graph node classification, we employ RGCN [45], GAT [9] and TransE [16] as in-graph node encoder. In the in-graph encoding procedure, RGCN and GAT are trained on node classification tasks, while TransE is trained on link prediction tasks.

For out-of-graph node classification, we provide neighbor information (neighbor and corresponding relation) of each out-of-graph node (in $\mathcal{V}_{\text{tra}}$, $\mathcal{V}_{\text{val}}$ or $\mathcal{V}_{\text{tst}}$). The target is first learning the out-of-graph node representation based on its neighbor information and in-graph node embedding, and then using out-of-graph node representation to predict the classes.

Evaluation Metrics: Following [46; 47], we use the Macro-F1 and accuracy as the node classification evaluation metrics. The formulas of the two metrics are as follows:

$$\text{Accuracy} = \frac{1}{n}\sum_{m=1}^{n} \mathbb{1}(\hat{y}_m = y_m),$$

where n is the number of samples, $\mathbb{1}(x)$ is the indicator function and y is out-of-graph node label.

$$\text{Macro-F1} = \frac{1}{c}\sum_{k=1}^{c} \frac{2 \cdot \text{precision}_k \cdot \text{recall}_k}{\text{precision}_k + \text{recall}_k},$$

where c is the number of classes, precision_k and recall_k means the precision or recall on the specific class k.

Hyper-parameter setting: Here we set the layer number of the in-graph transition module (L_{TI}) as 3. The layer number of out-of-graph transition module (L_{TO}) is set as 1 when out-of-graph node raw features are given. We choose ReLU as the activation function σ in our model. For proposed search algorithm, we initialize the model parameter learning rate as 0.003 and learning rate for probability distribution parameter as 0.01. We train the supernet for 30 epochs and sample 8 embedding extrapolating architectures in each epoch.

2) Overall performance comparison: Table II shows the out-of-graph node classification performance in the setting without raw features. We can see that proposed S2E consistently obtains the best performance, which demonstrates the effectiveness of the proposed embedding extrapolating framework and searching extrapolating architecture in S2E. Compared to S2E, baseline methods with fixed architecture are unable to handle diverse graph data, which is reflected in their results with different datasets and different in-graph node encoders. For instance, ConvL almost achieves the best performance in the dataset MobAPP when encoded by RGCN, while gets poor performance in the dataset ACM and IMDB. GAT obtains relatively good results in ACM when given in-graph node embedding encoded by GAT, but perform badly in the same dataset when encoded by RGCN. Moreover, we can see none of method have outstanding performance among baselines, as the second best results are achieved by different methods in different settings.

Among baseline methods, out-of-graph representation learning methods (MEAN, ConvL and GEN), which are specially designed to learn out-of-graph node representations, achieve better performance than GNN methods (GAT and GraphSAGE).

Experiment results in setting using raw features are shown in Table IV. It can be seen that S2E perform the best consistently, which shows that our design to search operation in $\mathcal{O}_{\text{TO}}$ is necessary. We can see that all the methods get better and stabler results than that in Table II (without raw features), which show that the raw features of out-of-graph nodes are beneficial to out-of-graph node classification tasks. Furthermore, in Table IV, we can observe that GNN methods (GAT and GraphSAGE) do not outperform MLP baseline when given raw features.

3) Searched architectures: Figure 3 and 4 show the searched architectures on different datasets when RGCN is used as in-graph node encoder. We can observe that different architectures are searched on different datasets and with different data inputs. Combining with the best performance in Table II, the necessity of designing data-specific architectures is obvious. Based on these searched architectures, we also find that there is always one `IDENTITY` operation in three in-graph node transition modules. As `IDENTITY` operation represents this transition operation layer is skipped, the searched architecture indicates that $L_{\text{TI}} = 2$ is the optimal choice for the in-graph node transition module. This conclusion can be verified in our ablation study on L_{TI} in Section V-D2.

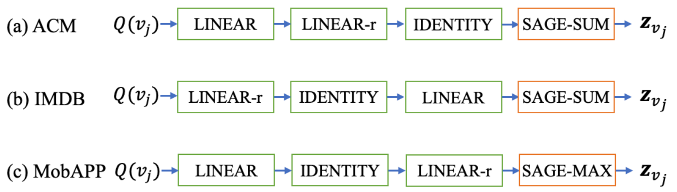

Fig. 3: Searched architectures in out-of-graph node classification without raw features.

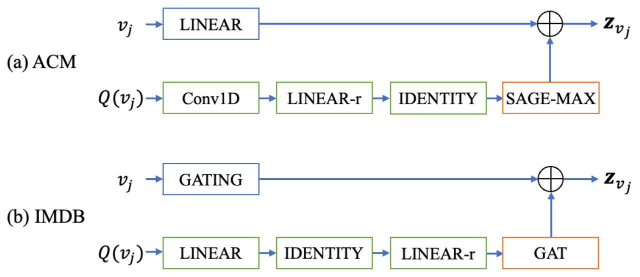

Fig. 4: Architecture searched by our algorithm in out-of-graph node classification using raw features.

B. Link Prediction

1) Experiment setting: Datasets: We perform the experiments of out-of-graph link prediction on two datasets (Table III) [12]: (1) FB15k-237 [48] is a knowledge graph dataset mainly contains Freebase entity pairs. (2) NELL-995 [49] mainly includes triplets that are collected by a lifelong learning system [50]. As link prediction datasets do not provide raw features, the out-of-graph link prediction experiments are all conducted in the setting without raw features.

Dataset processing and implementation: Similar as out-of-graph node classification, we split nodes each graph data into a in-graph node set ($\mathcal{V}_{\text{in}}$) and a out-of-graph node set ($\mathcal{V}_{\text{out}}$). We further divided out-of-graph node set ($\mathcal{V}_{\text{out}}$) into train, valid and test node sets ($\mathcal{V}_{\text{tra}}$, $\mathcal{V}_{\text{val}}$ and $\mathcal{V}_{\text{tst}}$).

As for in-graph node encoder, here RGCN [45], KB-GAT [51] (substitute GAT) and TransE [16] are used as in-

TABLE IV: Comparisons of different methods on out-of-graph tasks (with raw features).

Dataset	I-encoder	Metric	MLP	GAT	GraphSAGE	MEAN	ConvL	GEN	S2E
ACM	RGCN	**Macro-F1**	0.825 (0.004)	0.835 (0.012)	0.818 (0.013)	0.858 (0.006)	0.857 (0.007)	0.860 (0.004)	**0.875 (0.005)**
		Accuracy	0.828 (0.004)	0.834 (0.012)	0.817 (0.015)	0.859 (0.006)	0.858 (0.007)	0.861 (0.003)	**0.872 (0.004)**
	GAT	**Macro-F1**	0.824 (0.005)	0.826 (0.013)	0.828 (0.014)	0.885 (0.008)	0.887 (0.008)	0.879 (0.008)	**0.892 (0.005)**
		Accuracy	0.824 (0.004)	0.830 (0.017)	0.828 (0.015)	0.886 (0.007)	0.881 (0.008)	0.882 (0.007)	**0.890 (0.004)**
	TransE	**Macro-F1**	0.801 (0.006)	0.821 (0.015)	0.827 (0.014)	0.861 (0.006)	0.877 (0.007)	0.870 (0.006)	**0.873 (0.004)**
		Accuracy	0.821 (0.004)	0.829 (0.016)	0.828 (0.015)	0.871 (0.008)	0.871 (0.007)	0.877 (0.005)	**0.883 (0.004)**
IMDB	RGCN	**Macro-F1**	0.483 (0.003)	0.435 (0.015)	0.373 (0.016)	0.473 (0.007)	0.476 (0.007)	0.493 (0.014)	**0.514 (0.001)**
		Accuracy	0.506 (0.005)	0.448 (0.016)	0.449 (0.014)	0.505 (0.007)	0.503 (0.007)	0.529 (0.015)	**0.562 (0.005)**
	GAT	**Macro-F1**	0.482 (0.005)	0.414 (0.013)	0.377 (0.013)	0.467 (0.007)	0.481 (0.006)	0.494 (0.007)	**0.511 (0.004)**
		Accuracy	0.501 (0.006)	0.452 (0.014)	0.444 (0.013)	0.493 (0.006)	0.529 (0.007)	0.522 (0.005)	**0.558 (0.008)**
	TransE	**Macro-F1**	0.463 (0.004)	0.421 (0.015)	0.389 (0.013)	0.471 (0.008)	0.492 (0.007)	0.484 (0.015)	**0.510 (0.005)**
		Accuracy	0.512 (0.005)	0.463 (0.014)	0.442 (0.013)	0.496 (0.008)	0.510 (0.007)	0.509 (0.017)	**0.549 (0.004)**

graph node encoder. All of these encoders are trained on link prediction tasks.

In out-of-graph link prediction, each out-of-graph node have several related triplets. We sample a proportion (set as 50% in experiment) of triplets for training, which are used to learn the out-of-graph node representation. The rest of the triplets are used to evaluate the link prediction performance.

Evaluation metrics: We follow [16; 41; 42] to use the filtered ranking-based metrics for evaluation. We measure the rank of test triples among all possible subject or object substitutions. Denote the number of test triplets as n, and the rank of mth triplet prediction as rank_m. Here we consider the two metrics as follows:

$$\text{MRR} = \frac{1}{n}\sum_{m=1}^{n}\frac{1}{\text{rank}_m},$$
$$\text{Hit@k} = \frac{1}{n}\sum_{m=1}^{n}\mathbb{1}(\text{rank}_m \leq k).$$

Note that both of the ranking based evaluation metrics are non-differentiable.

Hyper-parameter setting: Here we set the layer number of the in-graph transition module (L_{TI}) as 3 and choose ReLU as the activation function σ in our model. In proposed search algorithm, we set model parameter learning rate as 0.001 and learning rate for probability distribution parameter as 0.003. We train the supernet for 50 epochs and sample 8 embedding extrapolating architectures in each epoch.

2) Overall performance comparison: The out-of-graph link prediction results are shown in Table V. It can be seen that the proposed S2E method consistently obtain the best performance, which shows the effectiveness of the embedding extrapolating framework and the architecture search algorithm in Section IV. Different from S2E, all baseline methods with fixed model architecture are unable to obtain stable and good performance in diverse real world graph learning tasks. For example, in experiment on dataset FB15k-237, GEN perform better when I-encoder is RGCN, while ConvL achieve better performance with I-encoder as KB-GAT. GAT achieves better performance than GraphSAGE when in-graph node encoder is RGCN and KB-GAT, while GraphSAGE performs better when in-graph node encoder is set as transE.

Among the baseline methods, we can see that the out-of-graph learning methods (MEAN, ConvL and GEN) generally perform better than baseline GNN methods (GAT and GraphSAGE).

3) Searched architectures: Figure 5 shows the searched architectures (top-1) by S2E when RGCN is in-graph node encoder. We can see that S2E obtains different architecture for different datasets and most of the transition operation candidates are used in the optimal architectures.

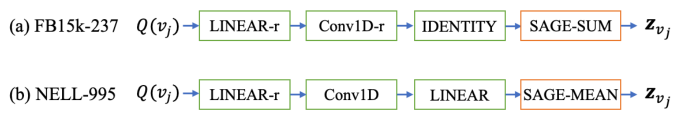

Fig. 5: Architectures searched by S2E in out-of-graph link prediction.

C. Understanding the Search Algorithm

In this section, we conduct experiments on the search efficiency and effectiveness of weight sharing strategy of our search algorithm to verify the reliability of proposed search algorithm. Experiments in this section are for out-of-graph node classification in the setting without raw features, and RGCN is used as an in-graph node encoder.

1) Search efficiency: To verify the search efficiency of the search algorithm, we compare the search time and classification performances of proposed method with architecture search baselines including Random Search and Bayesian Optimization, which are conducted on the search space introduced in Section IV-A. For Random and Bayesian search baselines, we

TABLE V: Comparisons of different methods on out-of-graph link prediction task (without raw features).

Dataset	I-encoder	Metric	GAT	GraphSAGE	MEAN	ConvL	GEN	S2E
FB15k-237	RGCN	**MRR**	0.199 (0.012)	0.171 (0.008)	0.247 (0.007)	0.231 (0.002)	<u>0.255</u> (0.007)	**0.274 (0.009)**
		Hit@1	0.125 (0.004)	0.102 (0.010)	0.159 (0.005)	0.161 (0.001)	<u>0.168</u> (0.001)	**0.199 (0.004)**
		Hit@3	0.211 (0.003)	0.179 (0.009)	0.260 (0.008)	0.245 (0.007)	<u>0.281</u> (0.003)	**0.314 (0.007)**
		Hit@10	0.351 (0.009)	0.314 (0.017)	<u>0.421</u> (0.008)	0.352 (0.006)	0.414 (0.012)	**0.459 (0.008)**
	KB-GAT	**MRR**	0.204 (0.005)	0.168 (0.012)	0.201 (0.007)	<u>0.220</u> (0.011)	0.219 (0.005)	**0.238 (0.003)**
		Hit@1	0.133 (0.001)	0.105 (0.004)	0.122 (0.003)	<u>0.144</u> (0.004)	0.140 (0.006)	**0.169 (0.002)**
		Hit@3	0.227 (0.009)	0.173 (0.010)	0.209 (0.011)	<u>0.241</u> (0.007)	0.245 (0.003)	**0.276 (0.010)**
		Hit@10	0.357 (0.014)	0.285 (0.023)	0.336 (0.027)	<u>0.362</u> (0.015)	0.352 (0.006)	**0.400 (0.013)**
	TransE	**MRR**	0.173 (0.012)	0.195 (0.007)	0.207 (0.011)	0.208 (0.004)	<u>0.209</u> (0.007)	**0.229 (0.004)**
		Hit@1	0.102 (0.005)	0.092 (0.004)	0.135 (0.007)	<u>0.146</u> (0.002)	0.134 (0.003)	**0.164 (0.004)**
		Hit@3	0.189 (0.004)	0.173 (0.001)	0.225 (0.003)	0.247 (0.006)	<u>0.248</u> (0.009)	**0.273 (0.003)**
		Hit@10	0.306 (0.019)	0.285 (0.017)	0.344 (0.023)	<u>0.364</u> (0.013)	0.354 (0.019)	**0.385 (0.009)**
NELL-995	RGCN	**MRR**	0.235 (0.009)	0.201 (0.009)	0.216 (0.009)	0.220 (0.007)	<u>0.242</u> (0.001)	**0.257 (0.006)**
		Hit@1	0.169 (0.003)	0.124 (0.005)	0.149 (0.006)	0.159 (0.004)	<u>0.179</u> (0.003)	**0.188 (0.002)**
		Hit@3	0.265 (0.006)	0.214 (0.006)	0.242 (0.008)	0.244 (0.007)	<u>0.275</u> (0.020)	**0.294 (0.005)**
		Hit@10	0.369 (0.017)	0.304 (0.014)	0.337 (0.008)	0.344 (0.013)	<u>0.369</u> (0.018)	**0.392 (0.011)**
	KB-GAT	**MRR**	0.160 (0.004)	0.148 (0.005)	0.188 (0.013)	0.173 (0.009)	<u>0.192</u> (0.004)	**0.217 (0.005)**
		Hit@1	0.095 (0.002)	0.090 (0.004)	0.097 (0.001)	0.101 (0.006)	<u>0.113</u> (0.006)	**0.130 (0.002)**
		Hit@3	0.179 (0.005)	0.170 (0.004)	0.199 (0.007)	0.193 (0.004)	<u>0.207</u> (0.007)	**0.226 (0.004)**
		Hit@10	0.297 (0.002)	0.261 (0.006)	<u>0.331</u> (0.005)	0.319 (0.009)	0.329 (0.010)	**0.350 (0.009)**
	TransE	**MRR**	0.135 (0.003)	0.140 (0.006)	0.172 (0.009)	0.166 (0.009)	<u>0.178</u> (0.002)	**0.191 (0.004)**
		Hit@1	0.079 (0.009)	0.088 (0.001)	<u>0.103</u> (0.004)	0.097 (0.002)	0.099 (0.004)	**0.110 (0.002)**
		Hit@3	0.150 (0.004)	0.161 (0.007)	<u>0.192</u> (0.006)	0.183 (0.009)	0.192 (0.007)	**0.206 (0.012)**
		Hit@10	0.247 (0.016)	0.259 (0.009)	0.289 (0.019)	0.267 (0.014)	<u>0.291</u> (0.006)	**0.330 (0.018)**

TABLE VI: Statistics of link prediction datasets. For out-of-graph nodes, we present the number of nodes in training/validation/test set.

Dataset	# In-graph Nodes	# Out-of-graph Nodes	# Relations
FB15k-237	11,541	600/600/1,800	237
NELL-995	75,492	400/400/1,200	200

sample 30 model architectures from the search space and one with the best validation performance is chosen to be trained from scratch, on top of which we obtain the test performance.

The results are shown in Table VII. As can be seen, both Random and Bayesian search take a long time to train model architectures. While the proposed searching extrapolating architecture only needs to train once on the search space to produce the optimum architecture. S2E consumes an order of magnitude less time compared to Random and Bayesian search algorithm. Although Random and Bayesian search spend more time searching, their classification performance is still not better than S2E, which verifies the search efficiency of S2E.

When setting transition layer number $L_{\text{TI}} = 3$ and $L_{\text{TO}} = 1$, the total number of candidate architectures is $6^3 \times 4 \times 4 = 3,456$. Although the number of architecture choices is not large, the total search time cost difference between the differentiable NAS methods with weight sharing (proposed searching extrapolating architecture) and traditional stand-alone NAS methods (Random and Bayesian which need to sample an architecture and train it from scratch) is still significant. Combining with the best performance in Table II and Table IV, the necessity of utilizing the differentiable search algorithm in proposed S2E is obvious.

2) Effectiveness of weight sharing: In Section IV-B, we use weight sharing to approximate the optimal of each single architecture. To show the effectiveness of the weight sharing, we visualize the rank correlation of the validation accuracy between the weight sharing strategy and the stand-alone approach on ACM dataset, as shown in Figure 6. For the stand-alone approach, we randomly select 50 model architectures, train and evaluate them from scratch. As for weight sharing, we inherit the corresponding subweights of the trained supernet for each structure in the architecture set and evaluate it.

As can be seen in Figure 6, there is a strong positive correlation between the rank of weight sharing validation

TABLE VII: Search time (in seconds) of running once each mode.

Search mode	**Random**		**Bayesian**		**S2E**	
Search Time/Acc.	Time	Acc.	Time	Acc.	Time	Acc.
ACM	3,792 (157)	0.716 (0.018)	5,089 (103)	0.726 (0.010)	270 (19)	0.739 (0.006)
IMDB	3,642 (131)	0.468 (0.026)	4,989 (162)	0.501 (0.017)	242 (16)	0.536 (0.004)
MobAPP	27,105 (1,882)	0.969 (0.012)	29,712 (1,482)	0.970 (0.006)	1,203 (93)	0.972 (0.005)

accuracy and the rank of stand-alone validation accuracy. And then, most selected structures that much higher validation ranks using weight sharing still perform better when using the strategy of stand-alone. This shows that the weight sharing strategy is capable of searching for appropriate structures in our formulated architecture search problem.

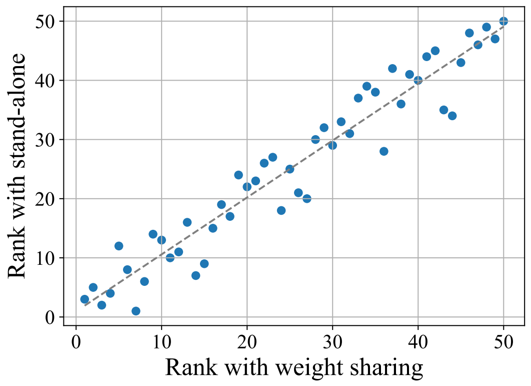

Fig. 6: Rank correlation between stand-alone and weight sharing approach on ACM dataset.

3) The necessity of non-differentiable evaluation metric: In link prediction task, the evaluation metrics we used are non-differentiable. To verify the necessity of these non-differentiable evaluation metrics, we substitute the evaluation metric into differentiable one (in this experiment we use accuracy) and directly optimize the architecture β instead of the probability distribution parameter. We use RGCN as in-graph node encoder in this part.

The results are shown in Table VIII. We can see that using non-differentiable evaluation metric perform better than directly using differentiable evaluation metric, which means that the differentiable evaluation metric cannot well express the optimization objectives for link prediction. The results also demonstrate that the proposed upper-level objective transformation in Section IV-B1 is necessary.

TABLE VIII: Link prediction MRR using different kinds of evaluation metrics to optimize architecture.

	non-differentiable metric (MRR)	differentiable metric (cross-entropy loss)
FB15k-237	**0.274**	0.249
NELL-995	**0.257**	0.231

D. Understanding the Search Space

In this section, we conduct experiment to verify the rationality of our designed search space on the proposed S2E for out-of-graph node classification tasks.

1) Effectiveness of the Designed Search Space: In this part, we conduct ablation study on three parts in our search space. Here we use RGCN as an in-graph node encoder. In experiment, we evaluate each module by fixing one of the operations ($\mathcal{C}_{\text{TO}} = \{\texttt{LINEAR}\}$, $\mathcal{C}_{\text{TI}} = \{\texttt{LINEAR-r}\}$ or $\mathcal{C}_{\text{AGG}} = \{\texttt{SAGE-SUM}\}$, respectively) and then search the other two modules as usual, to compare with our initial S2E method (**Full**). As can be seen from the results in Table IX, no matter which operation choice is fixed, the performance drop is observed, which demonstrates the effectiveness of designing these three modules when extrapolating embeddings for out-of-graph nodes. In the setting using raw features, we can see the largest performance decrease when we fix the aggregation module, demonstrating the importance of searching for a suitable aggregation module.

TABLE IX: Classification accuracy of Our method with reduced design spaces.

Setting	**Operation**	**ACM**	**IMDB**	**MobAPP**
Without raw features	$\mathcal{O}_{\text{TI}} = \{\texttt{LINEAR-r}\}$	0.702	0.532	0.969
	$\mathcal{O}_{\text{AGG}} = \{\texttt{SAGE-SUM}\}$	0.713	0.519	0.965
	Full	**0.739**	**0.536**	**0.972**
Using raw features	$\mathcal{O}_{\text{TI}} = \{\texttt{LINEAR-r}\}$	0.859	0.540	-
	$\mathcal{O}_{\text{AGG}} = \{\texttt{SAGE-SUM}\}$	0.822	0.512	-
	$\mathcal{O}_{\text{TO}} = \{\texttt{LINEAR}\}$	0.859	0.531	-
	Full	**0.872**	**0.562**	-

2) Influence of different number of transition layers: In this part, we conduct experiments to investigate how the performance of our S2E model is affected by different numbers of transition layers, using RGCN as the in-graph encoder. Here we delete the operation `IDENTITY` in the search space since it is equivalent to skip the transition layer. For in-graph transition module $\mathcal{O}_{\text{TI}}$, experiment results are shown in Table X. Our model achieves the best performance when the transition layer number is set as 2. For out-of-graph transition module $\mathcal{O}_{\text{TO}}$, our model obtains the best performance when the transition layer number is set as 1 (Table XI). Comparing the results in Table X and Table XI in the setting using raw features, we can see that the change of L_{TO} cause more performance changes, which shows the importance of raw features in out-of-graph node classification problem. When facing large transition layer numbers, we observe the dramatic performance degradation which might be caused by over-fitting.

TABLE X: Classification accuracy with different number of transition layers $\mathcal{O}_{\text{TI}}$.

Setting	L_{TI}	1	2	3	4
Without raw features	**ACM**	0.717	**0.718**	0.709	0.665
	IMDB	0.504	**0.519**	0.501	0.487
	MobAPP	0.968	**0.972**	0.968	0.959
Using raw features	**ACM**	0.853	**0.872**	0.851	0.829
	IMDB	0.543	**0.562**	0.543	0.524

TABLE XI: Classification accuracy with different number of transition layers $\mathcal{O}_{\text{TO}}$.

Setting	L_{TO}	1	2	3	4
Using raw features	**ACM**	**0.872**	0.796	0.757	0.711
	IMDB	**0.562**	0.508	0.481	0.473

VI. Conclusion

In this paper, we focus on handling embedding extrapolation for out-of-graph nodes, which is an important problem in real-world graphs. We first formulate the problem as architecture search to extrapolate embedding for out-of-graph nodes according to challenges including fixed in-graph node embedding and data diversity. Then we propose searching to extrapolate embedding (S2E) to handle the problem. Firstly, we introduce an embedding extrapolating framework to handle fixed in-graph node embedding. Based on that embedding extrapolating framework, we propose searching extrapolating architecture through stochastic relaxation to find suitable model architecture for a given real-world graph, which possesses the ability to tackle data diversity. To demonstrate the effectiveness of proposed method, we conduct extensive experiments for the out-of-graph node classification and link prediction problems in real-world datasets. The results show that proposed S2E has significant improvement compared to the baseline methods and solves aforementioned challenges well.

Appendix

1). Proof of Lemma 4.1

Let $\bar{\boldsymbol{\beta}} = \arg\max_{\boldsymbol{\beta}} \tilde{\mathcal{M}}(\bar{\boldsymbol{w}}, \boldsymbol{\beta})$. Obviously, we have

$$\tilde{\mathcal{M}}(\bar{\boldsymbol{w}}, \boldsymbol{\beta}) \leq \tilde{\mathcal{M}}(\bar{\boldsymbol{w}}, \bar{\boldsymbol{\beta}}).$$

Recall that $\mathcal{J}(\boldsymbol{\theta}) = \sum_{\boldsymbol{\beta}\in\mathcal{B}} \tilde{\mathcal{M}}(\bar{\boldsymbol{w}}, \boldsymbol{\beta}) p_{\boldsymbol{\theta}}(\boldsymbol{\beta})$, then

$$\begin{aligned}\mathcal{J}(\boldsymbol{\theta}) &\leq \sum_{\boldsymbol{\beta}\in\mathcal{B}} \big[\max_{\boldsymbol{\beta}} \tilde{\mathcal{M}}(\bar{\boldsymbol{w}}, \boldsymbol{\beta})\big] p_{\boldsymbol{\theta}}(\boldsymbol{\beta}), \\ &= \max_{\boldsymbol{\beta}} \tilde{\mathcal{M}}(\bar{\boldsymbol{w}}, \boldsymbol{\beta}) \sum_{\boldsymbol{\beta}\in\mathcal{B}} p_{\boldsymbol{\theta}}(\boldsymbol{\beta}) = \max_{\boldsymbol{\beta}} \tilde{\mathcal{M}}(\bar{\boldsymbol{w}}, \boldsymbol{\beta}),\end{aligned}$$

The equality in the first row holds only when $\tilde{\mathcal{M}}(\bar{\boldsymbol{w}}, \boldsymbol{\beta})$ is a constant as $\tilde{\mathcal{M}}(\bar{\boldsymbol{w}}, \bar{\boldsymbol{\beta}})$ or $p_{\boldsymbol{\theta}}(\bar{\boldsymbol{\beta}}) = 1$, hence $\sup_{\boldsymbol{\theta}} \mathcal{J}(\boldsymbol{\theta}) = \max_{\boldsymbol{\beta}} \tilde{\mathcal{M}}(\bar{\boldsymbol{w}}, \boldsymbol{\beta})$. Moreover, $\tilde{\mathcal{M}}(\bar{\boldsymbol{w}}, \boldsymbol{\beta})$ is not constant in our problem, thus $p_{\bar{\boldsymbol{\theta}}}(\bar{\boldsymbol{\beta}}) = 1$, which end the proof.

2). Complete update procedure of $\boldsymbol{W}$

The update procedure of $\boldsymbol{W}$ is in Algorithm 2. We first calculate and save the gradient of each parts in $\boldsymbol{W}(\boldsymbol{\beta}_i)$ of sampled architecture from the supernet. Then we accumulate the gradients of each module in the supernet and update the supernet.

Algorithm 2 The update scheme of $\boldsymbol{W}$

```
Require: $\boldsymbol{W}$, sampled architectures $\boldsymbol{\beta}_1, \boldsymbol{\beta}_2, ..., \boldsymbol{\beta}_k$.
 1: for $i = 1$ to $k$ do
 2:     Calculate and save the gradient of each parts in $\boldsymbol{W}(\boldsymbol{\beta}_i)$
        as (7)
 3: end for
 4: for $\tau = 1$ to $L_{\text{TI}}$ do
 5:     Update $w_{\text{TI}}^{\tau}$ like (8)
 6: end for
 7: for $\tau = 1$ to $L_{\text{TO}}$ do
 8:     Update $w_{\text{TO}}^{\tau}$ like (8)
 9: end for
10: Update $w_{\text{AGG}}$ like (8)
11: return The updated $\boldsymbol{W}$.
```

References

[1] R. Zafarani, M. A. Abbasi, and H. Liu, *Social media mining: an introduction*. Cambridge University Press, 2014.

[2] Y. Ye, S. Hou, L. Chen, J. Lei, W. Wan, J. Wang, Q. Xiong, and F. Shao, "Out-of-sample node representation learning for heterogeneous graph in real-time android malware detection," in *Joint Conference on Artificial Intelligence*, 2019.

[3] F. Radicchi, S. Fortunato, and A. Vespignani, "Citation networks," *Models of Science Dynamics: Encounters Between Complexity Theory and Information Sciences*, pp. 233–257, 2012.

[4] T. Paek, M. Gamon, S. Counts, D. Chickering, and A. Dhesi, "Predicting the importance of newsfeed posts and social network friends," in *AAAI Conference on Artificial Intelligence*, vol. 24, no. 1, 2010, pp. 1419–1424.

[5] Y. Hei, R. Yang, H. Peng, L. Wang, X. Xu, J. Liu, H. Liu, J. Xu, and L. Sun, "Hawk: Rapid android malware detection through heterogeneous graph attention networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[6] M. Gönen, "Predicting drug–target interactions from chemical and genomic kernels using bayesian matrix factorization," *Bioinformatics*, vol. 28, no. 18, pp. 2304–2310, 2012.

[7] M. Tripathy, S. Champati, and H. K. Bhuyan, "Knowledge discovery in a recommender system: The matrix factorization approach," *Journal of Information & Knowledge Management*, vol. 21, no. 04, p. 2250051, 2022.

[8] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," *Advances in Neural Information Processing Systems*, 2017.

[9] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations*, 2018.

[10] T. Hamaguchi, H. Oiwa, M. Shimbo, and Y. Matsumoto, "Knowledge transfer for out-of-knowledge-base entities: a graph neural network approach," in *International Joint Conference on Artificial Intelligence*, 2017, pp. 1802–1808.

[11] P. Wang, J. Han, C. Li, and R. Pan, "Logic attention based neighborhood aggregation for inductive knowledge graph embedding," in *AAAI Conference on Artificial Intelligence*, 2019, pp. 7152–7159.

[12] J. Baek, D. B. Lee, and S. J. Hwang, "Learning to extrapolate knowledge: Transductive few-shot out-of-graph link prediction," *Advances in Neural Information Processing Systems*, pp. 546–560, 2020.

[13] S. Bhagat, G. Cormode, and S. Muthukrishnan, "Node classification in social networks," *Social Network Data Analytics*, pp. 115–148, 2011.

[14] M. Nickel, K. Murphy, V. Tresp, and E. Gabrilovich, "A review of relational machine learning for knowledge graphs," *Proceedings of the IEEE*, pp. 11–33, 2015.

[15] F. Tian, B. Gao, Q. Cui, E. Chen, and T.-Y. Liu, "Learning deep representations for graph clustering," in *AAAI Conference on Artificial Intelligence*, 2014.

[16] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," *Advances in Neural Information Processing Systems*, 2013.

[17] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2014, pp. 701–710.

[18] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *International Conference on Learning Representations*, 2016.

[19] P. Jain, Z. Wu, M. Wright, A. Mirhoseini, J. E. Gonzalez, and I. Stoica, "Representing long-range context for graph neural networks with global attention," *Advances in Neural Information Processing Systems*, 2021.

[20] Z. Bi, T. Zhang, P. Zhou, and Y. Li, "Knowledge transfer for out-of-knowledge-base entities: Improving graph-neural-network-based embedding using convolutional layers," *IEEE Access*, pp. 159 039–159 049, 2020.

[21] X. He, K. Zhao, and X. Chu, "Automl: A survey of the state-of-the-art," *Knowledge-Based Systems*, vol. 212, p. 106622, 2021.

[22] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *International Conference on Learning Representations*, 2017.

[23] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, "Large-scale evolution of image classifiers," in *International Conference on Machine Learning*. PMLR, 2017, pp. 2902–2911.

[24] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 4780–4789.

[25] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *International Conference on Machine Learning*. PMLR, 2018, pp. 4095–4104.

[26] X. Ning, C. Tang, W. Li, Z. Zhou, S. Liang, H. Yang, and Y. Wang, "Evaluating efficient performance estimators of neural architectures," *Advances in Neural Information Processing Systems*, vol. 34, pp. 12 265–12 277, 2021.

[27] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," in *International Conference on Learning Representations*, 2018.

[28] S. Xie, H. Zheng, C. Liu, and L. Lin, "Snas: stochastic neural architecture search," in *International Conference on Learning Representations*, 2018.

[29] Y. Akimoto, S. Shirakawa, N. Yoshinari, K. Uchida, S. Saito, and K. Nishida, "Adaptive stochastic natural gradient method for one-shot neural architecture

search," in *International Conference on Machine Learning*. PMLR, 2019, pp. 171–180.

[30] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," in *International Conference on Machine Learning*. PMLR, 2017, pp. 1263–1272.

[31] Y. Gao, H. Yang, P. Zhang, C. Zhou, and Y. Hu, "Graph neural architecture search," in *International Conference on International Joint Conferences on Artificial Intelligence*, 2021, pp. 1403–1409.

[32] K. Zhou, Q. Song, X. Huang, and X. Hu, "Auto-gnn: Neural architecture search of graph neural networks," *arXiv preprint arXiv:1909.03184*, 2019.

[33] H. Zhao, Q. Yao, and W. Tu, "Search to aggregate neighborhood for graph neural network," in *International Conference on Data Engineering*. IEEE, 2021, pp. 552–563.

[34] Y. Li, Z. Wen, Y. Wang, and C. Xu, "One-shot graph neural architecture search with dynamic search space," in *AAAI Conference on Artificial Intelligence*, 2021, pp. 8510–8517.

[35] W. Zhang, Z. Lin, Y. Shen, Y. Li, Z. Yang, and B. Cui, "Dfg-nas: Deep and flexible graph neural architecture search," *arXiv preprint arXiv:2206.08582*, 2022.

[36] W. Zhang, Y. Shen, Z. Lin, Y. Li, X. Li, W. Ouyang, Y. Tao, Z. Yang, and B. Cui, "Pasca: A graph neural architecture search system under the scalable paradigm," in *Proceedings of the ACM Web Conference 2022*, 2022, pp. 1817–1828.

[37] Y. Ding, Q. Yao, H. Zhao, and T. Zhang, "Diffmg: Differentiable meta graph search for heterogeneous graph neural networks," in *ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, 2021, pp. 279–288.

[38] Z. Wang, H. Zhao, and C. Shi, "Profiling the design space for graph neural networks based collaborative filtering," in *ACM International Conference on Web Search and Data Mining*, 2022, pp. 1109–1119.

[39] Z. Wang, S. Di, and L. Chen, "Autogel: An automated graph neural network with explicit link information," *Advances in Neural Information Processing Systems*, pp. 24 509–24 522, 2021.

[40] Y. Zhang, Q. Yao, W. Dai, and L. Chen, "Autosf: Searching scoring functions for knowledge graph embedding," in *International Conference on Data Engineering*. IEEE, 2020, pp. 433–444.

[41] T. Trouillon, J. Welbl, S. Riedel, É. Gaussier, and G. Bouchard, "Complex embeddings for simple link prediction," in *International Conference on Machine Learning*, 2016, pp. 2071–2080.

[42] Z. Sun, Z. Deng, J. Nie, and J. Tang, "Rotate: Knowledge graph embedding by relational rotation in complex space," in *International Conference on Learning Representations*, 2019.

[43] K. Ahmed and L. Torresani, "Maskconnect: Connectivity learning by gradient descent," in *Proceedings of the European Conference on Computer Vision*, 2018, pp. 349–365.

[44] S. Shirakawa, Y. Iwata, and Y. Akimoto, "Dynamic optimization of neural network structures using probabilistic modeling," in *AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[45] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *European Semantic Web Conference*. Springer, 2018, pp. 593–607.

[46] X. Wang, H. Ji, C. Shi, B. Wang, Y. Ye, P. Cui, and P. S. Yu, "Heterogeneous graph attention network," in *World Wide Web conference*, 2019, pp. 2022–2032.

[47] C. Zhang, D. Song, C. Huang, A. Swami, and N. V. Chawla, "Heterogeneous graph neural network," in *ACM SIGKDD International Conference on Knowledge Discovery & Data mining*, 2019, pp. 793–803.

[48] K. Toutanova, D. Chen, P. Pantel, H. Poon, P. Choudhury, and M. Gamon, "Representing text for joint embedding of text and knowledge bases," in *Proceedings of the 2015 conference on empirical methods in natural language processing*, 2015, pp. 1499–1509.

[49] W. Xiong, T. Hoang, and W. Y. Wang, "Deeppath: A reinforcement learning method for knowledge graph reasoning," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017, pp. 564–573.

[50] T. Mitchell, W. Cohen, E. Hruschka, P. Talukdar, B. Yang, J. Betteridge, A. Carlson, B. Dalvi, M. Gardner, B. Kisiel *et al.*, "Never-ending learning," *Communications of the ACM*, vol. 61, no. 5, pp. 103–115, 2018.

[51] D. Nathani, J. Chauhan, C. Sharma, and M. Kaul, "Learning attention-based embeddings for relation prediction in knowledge graphs," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, 2019, pp. 4710–4723.