A Good Neighbor, A Found Treasure: Mining Treasured Neighbors for Knowledge Graph Entity Typing

Zhuoran Jin[1,2], **Pengfei Cao**[1,2], **Yubo Chen**[1,2], **Kang Liu**[1,2,3], **Jun Zhao**[1,2]
[1] School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing, China
[2] National Laboratory of Pattern Recognition, Institute of Automation, CAS, Beijing, China
[3] Beijing Academy of Artificial Intelligence, Beijing, China
{zhuoran.jin, pengfei.cao, yubo.chen, kliu, jzhao}@nlpr.ia.ac.cn

Abstract

The task of knowledge graph entity typing (KGET) aims to infer the missing types for entities in knowledge graphs. Some pioneering work has proved that neighbor information is essential for the task. However, existing methods only leverage the one-hop neighbor information of the central entity, ignoring the multi-hop neighbor information that can provide valuable clues for inference. Besides, we also observe that there are co-occurrence relations between types, which is very helpful in alleviating the false-negative problem. In this paper, we propose a novel method called **Mi**ning Treasured **Ne**ighbo**r**s (**MiNer**) to make use of these two characteristics. Firstly, we devise a *Neighbor Information Aggregation* module to aggregate the neighbor information. Then, we propose an *Entity Type Inference* module to mitigate the adverse impact of the irrelevant neighbor information. Finally, a *Type Co-occurrence Regularization* module is designed to prevent the model from overfitting the false-negative examples caused by missing types. Experimental results on two widely used datasets indicate that our approach significantly outperforms previous state-of-the-art methods.[1]

1 Introduction

Knowledge graphs (KGs) store huge amounts of structured data in the form of triples, i.e., (head entity, relation, tail entity). Each entity in KGs is labeled with one or more types. As shown in Figure 1, the entity *Einstein* not only belongs to *Scientist* type, but also *Physicist* type. The entity type information is very important and can benefit many natural language processing (NLP) applications, such as entity linking (Chen et al., 2018), relation extraction (Vashishth et al., 2018), knowledge graph embedding (Xie et al., 2016) and text generation (Dong et al., 2021).

[1] https://github.com/jinzhuoran/MiNer/

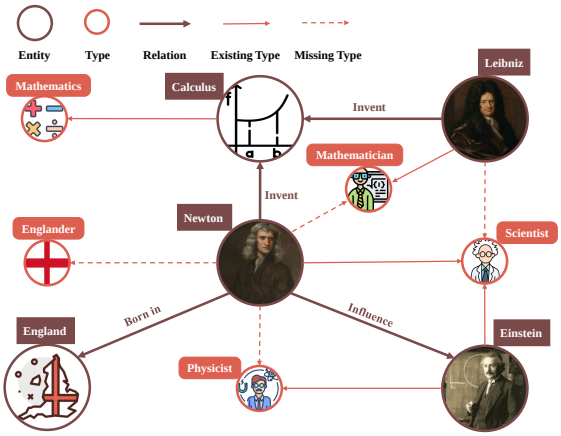

Figure 1: An example of a KG fragment. Large brown circles denote entities and small red circles denote types. Brown solid lines denote relations between entities, red solid lines denote existing types of entities and red dotted lines denote missing types of entities.

Unfortunately, KGs usually suffer from entity type incompleteness problems. More specifically, one entity may have multiple types, while the annotated entity type information is usually incomplete. Take Figure 1 as an example, the entity *Newton* should be labeled with *Scientist*, *Mathematician*, *Physicist* and *Englander* types. However, only the *Scientist* type is annotated in the KG. According to the statistics on the FB15k (Moon et al., 2017), 10% of entities have the */music/artist* type, but missing the */people/person* type, which indicates the type incompleteness problem is not negligible. Therefore, we focus on knowledge graph entity typing (KGET), which aims to infer the missing types from existing types for entities in KGs.

Great efforts have been devoted to tackling the KGET task, which can be mainly divided into embedding-based methods (Moon et al., 2017; Zhao et al., 2020) and graph neural network-based methods (Pan et al., 2021; Zhuo et al., 2022). Embedding-based methods focus on learning low-dimensional vector representations of entities, rela-

Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, pages 480 - 490
December 7-11, 2022

tions and entity types, then predict missing types based on a scoring function. Although embedding-based methods are simple and intuitive, they ignore the rich neighbor information of entities. By contrast, graph neural network-based methods effectively leverage the neighbor information by modeling it as the graph-structured data to infer the missing types, which has shown to be the most effective for the KGET task (Pan et al., 2021). Despite these successful efforts, existing methods ignore the *Multi-hop Neighbor Information*, and *Type Co-occurrence Information*, which is very important for the KGET task.

Multi-hop neighbor information can provide more valuable clues for inference. For example in Figure 1, based on the one-hop neighbor information alone, it is difficult to predict the entity *Newton* has *Mathematician* type. Fortunately, multi-hop neighbor information can provide more conclusive clues. For instance, there are facts that *Newton* and *Leibniz* (two-hop neighbor) both invent *Calculus*, and *Leibniz* has *Mathematician* type. Combining the two, the model can easily conclude *Newton* has *Mathematician* type. Nevertheless, aggregating multi-hop neighbors may bring irrelevant information. As shown in Figure 1, only the triple (*Newton*, *Born in*, *England*) plays a decisive role in inferring the *Englander* type for *Newton*, while others contribute less. Therefore, the first challenging problem is how to mine treasured neighbor information for inference.

Type co-occurrence information can facilitate alleviating the false-negative problem. Some entity types should have been stored in KGs, but they are not marked. Most existing methods simply treat them as negative samples. Therefore, they face a serious false-negative problem that affects model performance. We observe a rich amount of type co-occurrence information in KGs, which can be used to address the false-negative problem. For instance, both *Scientist* type and *Physicist* type often go hand in hand. However, *Scientist* type and *Actor* type rarely belong to the same entity. If the model makes use of the prior knowledge, it can alleviate the memorization of false labels and accurately predict the missing types. Therefore, the second challenging problem is how to leverage the type co-occurrence information.

In this paper, we propose a novel method termed as **Mi**ning Treasured **Ne**ighbo**r**s (**MiNer**) to address aforementioned problems. The proposed method consists of three modules: *Neighbor Information Aggregation* module, *Entity Type Inference* module and *Type Co-occurrence Regularization* module. First, the *Neighbor Information Aggregation* module aims to aggregate more neighbor information, including one-hop neighbor information and multi-hop neighbor information. This module can generalize to any number of hops. To mitigate the adverse impact of the irrelevant neighbor information, the *Entity Type Inference* module mines the valuable neighbor information for central entities via two ways: type-specific local inference and type-agnostic global inference. In addition, we leverage the type co-occurrence information to alleviate the memorization of the false labels. We propose *Type Co-occurrence Regularization* module to correct false negative examples caused by missing types. Experimental results on two widely used datasets indicate that our approach significantly outperforms previous state-of-the-art methods.

Our contributions are summarized as follows:

- We propose a novel method called MiNer, designed to aggregate both one-hop and multi-hop neighbors, then mine valuable information for missing type inference.
- We notice the strong correlations between different types and propose type co-occurrence regularization to mitigate the impact of the false-negative problem.
- We conduct thorough experiments with ablation studies on two widely used datasets, demonstrating our approach significantly outperforms previous state-of-the-art methods.

2 Related Work

Knowledge graph entity typing (KGET) is an essential sub-task of knowledge graph completion (KGC) that has been researched for decades. Existing approaches for the task can be mainly divided into embedding-based methods and graph neural network-based methods.

Embedding-based Methods. The entities with known types can be treated as as special triples with a unique relation "Has type", e.g., (*Newton*, *Has type*, *Physicist*). In this way, the KGET can be formulated as a link prediction task. Existing knowledge graph embedding (KGE) methods (Bordes et al., 2013; Wang et al., 2014; Trouillon et al., 2016; Sun et al., 2019; Chao et al., 2021) can be

used to infer the missing types of *Newton* by completing the triple (*Newton*, *Has type*, ?).

Although using KGE methods can address the KGET task to some degree, it lacks the diversity of relation types. Therefore, Moon et al. (2017) propose the ETE model to tackle this problem. ETE first learns entity embeddings and relation embeddings with a KGE model on KGs that do not have entity types, then trains the embedding of each entity to be closer to the embedding of its type. For better expressing and reasoning capability, Zhao et al. (2020) propose the ConnectE model, which considers both local entity typing information and global triple knowledge in KGs. ConnectE first uses TransE (Bordes et al., 2013) to obtain the entity embeddings, then infers the missing types according to two inference mechanisms. One is E2T mechanism that focuses on mapping entities from entity space to entity types space. Another is TRT mechanism, which is based on the assumption that the relation can remain unchanged when replacing the entities in the triple with their types.

Graph Neural Network-based Methods. Although embedding-based methods are simple and intuitive, they ignore the rich neighbor information. Graph neural network (GNN) has been proved to be quite successful in modeling graph-structured data. Considering that KG is also a kind of graph-structured data, existing GNN methods, such as R-GCN (Schlichtkrull et al., 2018), GAT (Veličković et al., 2018), WGCN (Shang et al., 2019) and CompGCN (Vashishth et al., 2020), can be used to aggregate the neighbor information better.

Pan et al. (2021) argue this may introduce irrelevant information as noise and affect the performance of entity typing. Pan et al. (2021) propose the CET, which fully utilizes the neighbor information according to N2T mechanism and Agg2T mechanism. N2T mechanism can independently use each neighbor to infer the missing types of central entities. Agg2T mechanism is designed to aggregate neighbors to infer the missing types. Zhuo et al. (2022) present the AttEt to capture the different weight distribution of the fine-grained entity types on each neighbor.

Despite the vast progress that the KGET task has made in recent years, existing methods only leverage the one-hop neighbor information, and ignore the multi-hop neighbor information and type co-occurrence information that is very important for the KGET task.

Other Methods. Zhao et al. (2022) propose a multiplex relational graph attention network as the encoder to learn embeddings and then use ConnectE as the decoder to make entity type inference. There are also some methods (Neelakantan and Chang, 2015; Jin et al., 2018, 2019) that use additional information (i.e., entity name, text description, and property) to infer the missing types.

3 Task Definition

Formally, we consider a KG $\mathcal{G}$ containing the triples in the form of $(e, r, \tilde{e})$ and the entity type information in the form of (e, t), where $e, \tilde{e} \in \mathcal{E}$, $r \in \mathcal{R}$, $t \in \mathcal{T}$, and $\mathcal{E}, \mathcal{R}, \mathcal{T}$ are the entity set, relation set and entity type set, respectively. One entity may have multiple types, while the annotated entity type is usually incomplete. The neighborhoods of entities can provide more valuable and richer information for inferring missing types. Following the setting of Pan et al. (2021), we also regard the existing types of each entity as its one-hop neighbors. Therefore, this paper aims to infer the missing types by considering the neighbor information (i.e., one-hop neighbor information and multi-hop neighbor information) of entities.

4 Method

Our approach is shown in Figure 2, which consists of three primary components: (1) *Neighbor Information Aggregation*, which aggregates the information from one-hop neighbors and multi-hop neighbors; (2) *Entity Type Inference*, which mines valuable neighbor information for inferring missing types; and (3) *Type Co-occurrence Regularization*, which prevents the model from overfitting false-negative samples by using type co-occurrence information. We will detail these three modules as follows.

4.1 Neighbor Information Aggregation

As mentioned above, it is not enough to solely use one-hop neighbors, multi-hop neighbors are also important. Neighbor information aggregation module aims to aggregate the two kinds of neighbor information.

One-hop Neighbor Aggregation. The one-hop neighbors of an entity are the most straightforward treasure for inferring the entity types. We first unify the one-hop outgoing neighbors and incoming neighbors as the one-hop neighbors of the central entity. Then, we follow the translational

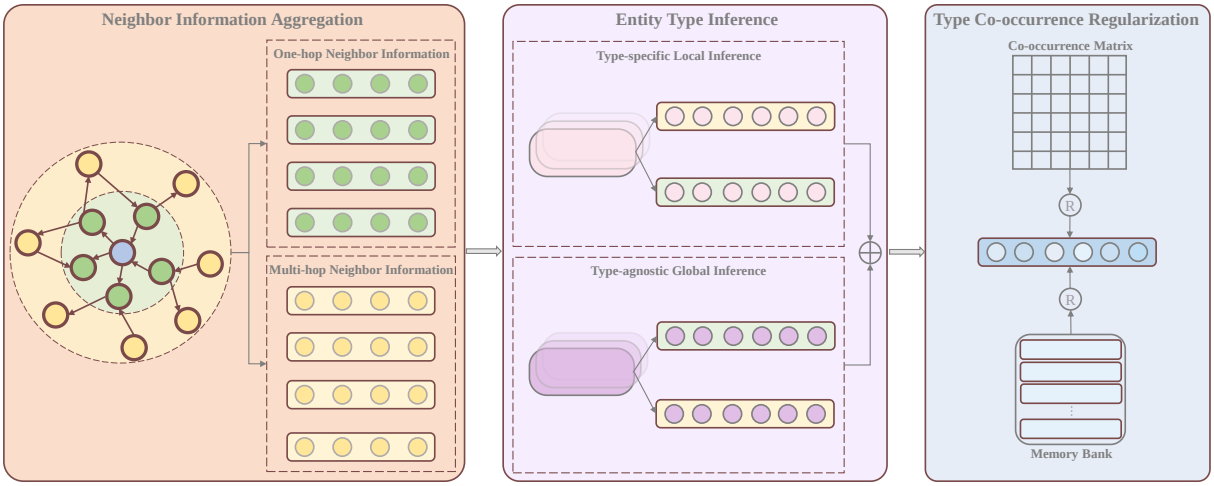

Figure 2: The main architecture of MiNer, consists of three primary modules. The blue circle denotes the central entity, the green circles denote the one-hop neighbors, the yellow circles denotes the multi-hop neighbors.

assumption of TransE (Bordes et al., 2013) to obtain the one-hop neighbor information. We choose TransE for its simplicity and efficiency. Formally, for the central entity e, its representation aggregated from one-hop neighbor $(r, \tilde{e})$[2] can be computed as follows:

$$\boldsymbol{h}_{(r,\tilde{e}),1} = \tilde{\boldsymbol{e}} - \mathbb{I}(\boldsymbol{r}), \tag{1}$$

where $(e, r, \tilde{e})$ is a triple in the KG $\mathcal{G}$. $\tilde{\boldsymbol{e}}$ and $\boldsymbol{r}$ denote the representations of the entity $\tilde{e}$ and relation r, respectively. $\mathbb{I}(\cdot)$ is a function that equals to 1 if $(e, r, \tilde{e}) \in \mathcal{G}$, or equals to -1 if $(\tilde{e}, r, e) \in \mathcal{G}$. $\boldsymbol{h}_{(r,\tilde{e}),1} \in \mathbb{R}^d$ is the representation aggregated from the one-hop neighbor $(r, \tilde{e})$.

Multi-hop Neighbor Aggregation. In addition to one-hop neighbors, we also consider multi-hop neighbors, which can provide valuable inference evidence. For multi-hop neighbor aggregation, the main idea is: we iteratively represent the $(n-1)$-hop neighbors by the n-hop neighbors and then represent the central entity by its one-hop neighbors. We first take two-hop neighbors as an example to illustrate the aggregation process, and then generalize it to the case of more hops. Formally, for the central entity e, its representation aggregated from its two-hop neighbor can be computed as follows:

$$\boldsymbol{h}_{(r,\tilde{e}),2} = \mathcal{M}_2(\tilde{e}) - \mathbb{I}(\boldsymbol{r}), \tag{2}$$

where $\boldsymbol{h}_{(r,\tilde{e}),2} \in \mathbb{R}^d$ is the representation aggregated from two-hops neighbor via the one-hop neighbor $(r, \tilde{e})$. $\mathcal{M}_2(\cdot)$ is used to compute the one-hop neighbor representation via two-hop neighbor representations of the central entity. Its calculation can be defined as follows:

$$\mathcal{M}_2(e) = \frac{1}{|\mathcal{N}(e)|} \sum_{(r_i,\tilde{e}_i)\in\mathcal{N}(e)} (\tilde{\boldsymbol{e}}_i - \mathbb{I}(\boldsymbol{r}_i)), \tag{3}$$

where $\mathcal{N}(e)$ denotes the one-hop neighbors of the entity e. To generalize our method, we consider aggregating the multi-hop neighbor information within h ($h \geq 3$) hops, which can be computed as follows:

$$\begin{aligned} \boldsymbol{h}_{(r,\tilde{e}),h} &= \mathcal{M}_h(\tilde{e}) - \mathbb{I}(\boldsymbol{r}), \\ \mathcal{M}_h(e) &= \frac{1}{|\mathcal{N}(e)|} \sum_{(r_i,\tilde{e}_i)\in\mathcal{N}(e)} (\rho_1(\tilde{e}_i) - \mathbb{I}_i(\boldsymbol{r}_i)), \\ \rho_j(e) &= \alpha_j \boldsymbol{e} + \frac{1-\alpha_j}{|\mathcal{N}(e)|} \sum_{(r_i,\tilde{e}_i)\in\mathcal{N}(e)} (\rho_{j+1}(\tilde{e}_i) - \mathbb{I}_i(\boldsymbol{r}_i)), \end{aligned} \tag{4}$$

where $\mathcal{M}_h(\cdot)$ denotes to aggregate the h-hop neighbor information, $\rho_j(\cdot)$ is calculated by the skip connection of j-hop neighbors and $(j+1)$-hop neighbors, and $\rho_j(e) = \boldsymbol{e}$ when $j = h-2$.

4.2 Entity Type Inference

In fact, different neighbors have different effects on different types of the central entity. For example, the *Englander* type of *Newton* can only be indicated by a few neighbors (i.e., *England*), and most of the neighbors (i.e., *Einstein* and *Leibniz*) are irrelevant. Therefore, we need to mitigate the adverse effect of the useless neighbor information. Inspired by the class-specific residual attention (CSRA) (Zhu and Wu, 2021) that captures different spatial regions occupied by objects from different categories, we

[2]For outgoing neighbors (i.e., $(e, r, \tilde{e})$) and incoming neighbors (i.e., $(\tilde{e}, r, e)$), we both denote $(r, \tilde{e})$ as the one-hop neighbor of the central entity e.

propose the entity type inference module to capture the accurate and useful neighbor features. We first use the non-linear classifier to compute the score vector $\boldsymbol{s}_i$ for i-th neighbor $(r_i, \tilde{e}_i)$:

$$\boldsymbol{s}_i = \boldsymbol{W}(\sigma(\boldsymbol{h}_i)), \tag{5}$$

where $\boldsymbol{W} \in \mathbb{R}^{|\mathcal{T}| \times d}$ is the weight matrix. $|\mathcal{T}|$ is the total number of entity types. $\sigma(\cdot)$ is the activation function (e.g., ReLU). $\boldsymbol{h}_i = [h_i^1, h_i^2, \ldots, h_i^d]^T \in \mathbb{R}^d$ is the hidden representation of the central entity aggregated from the i-th neighbor, $\boldsymbol{s}_i = [s_i^1, s_i^2, \ldots, s_i^{|\mathcal{T}|}]^T \in \mathbb{R}^{|\mathcal{T}|}$ is the score vector of the i-th neighbor, s_i^j indicates the probability score for inferring j-th type based on the neighbor $(r_i, \tilde{e}_i)$. According to the score vector for each neighbor, we predict the central entity types by type-specific local inference and type-agnostic global inference.

Type-specific Local Inference. Since different neighbors have different effects on different types of central entity, we devise a type-specific attention. Formally, for the central entity e, we define the type-specific attention weight a_i^j for its i-th neighbor $(r_i, \tilde{e}_i)$ and j-th type as:

$$a_i^j = \frac{\exp\left(s_i^j / T\right)}{\sum_{(r_k, \tilde{e}_k) \in \mathcal{N}(e)} \exp\left(s_k^j / T\right)}, \tag{6}$$

where T is the temperature controlling the weight's sharpness, a_i^j indicates the importance of i-th neighbor for inferring j-th type. Then, we can compute the type-specific local score of j-th type:

$$n^j = \sum_{(r_k, \tilde{e}_k) \in \mathcal{N}(e)} a_i^j s_k^j. \tag{7}$$

Therefore, we can represent the type-specific local score vector for the central entity e as $\boldsymbol{n} = [n^1, n^2, \ldots, n^{|\mathcal{T}|}]^T$.

Type-agnostic Global Inference. If two entities have similar types, their hidden representations should be close. It is necessary to represent the entities well in semantic space. According to the vanilla GCN (Kipf and Welling, 2017), we encode the central entity e as the average pooling of the hidden representations of its neighbors. Therefore, the type-agnostic score can be computed as follows:

$$\boldsymbol{g} = \frac{1}{|\mathcal{N}(e)|} \sum_{(r_k, \tilde{e}_k) \in \mathcal{N}(e)} \boldsymbol{s}_k, \tag{8}$$

where $\boldsymbol{g}$ means the type-agnostic global score vector of the entity e.

Type Probability Prediction. We combine the type-specific local score and type-agnostic global score together to get the mixed score $\boldsymbol{u}$:

$$\boldsymbol{u} = \beta_1 \boldsymbol{n} + \beta_2 \boldsymbol{g}, \tag{9}$$

where β_1 and β_2 are hyper-parameters for balance. According to CSRA (Zhu and Wu, 2021), we use the multi-head attention mechanism to compute the final score $\boldsymbol{f}$:

$$\boldsymbol{f} = \sum_{i=1}^{H} \boldsymbol{u}_{T_i}, \tag{10}$$

where H is the number of attention heads, $\boldsymbol{u}_{T_i}$ is the mixed score at temperature T_i. We predict the type probability $\boldsymbol{p} = [p_1, \ldots, p_{|\mathcal{T}|}]^T \in \mathbb{R}^{|\mathcal{T}|}$ based on both one-hop and multi-hop neighbors:

$$\boldsymbol{p} = \phi(\lambda \boldsymbol{f}^1 + (1 - \lambda)\boldsymbol{f}^h), \tag{11}$$

where $\boldsymbol{f}^1 \in \mathbb{R}^{|\mathcal{T}|}$ means the one-hop neighbors' final score and $\boldsymbol{f}^h \in \mathbb{R}^{|\mathcal{T}|}$ means the multi-hop neighbors' final score, λ is a hyper-parameter, ϕ denotes the sigmoid activation function.

4.3 Type Co-occurrence Regularization

As mentioned above, the type co-occurrence information is the overlooked treasure. Meanwhile, due to the missing of partial entity types, simply regarding these missing types as negative types will lead to false-negative samples in the training data. According to early learning phenomenon (Arpit et al., 2017), the model will first fit the training data with clean labels during an early learning phase, then memorize the training data with false labels. Inspired by early learning regularization (Liu et al., 2020), we propose type co-occurrence regularization, which leverages the type co-occurrence statistical information to alleviate the memorization of the false labels:

$$\mathcal{R}_{TCR} = \frac{1}{|\mathcal{E}|} \sum_{i=1}^{|\mathcal{E}|} \log\left(1 - \langle \mathcal{S}(\boldsymbol{p}_i(k)), \boldsymbol{t}_i(k) \rangle\right), \tag{12}$$

where $\boldsymbol{p}_i(k) \in \mathbb{R}^{|\mathcal{T}|}$ and $\boldsymbol{t}_i(k) \in \mathbb{R}^{|\mathcal{T}|}$ denote the i-th entity's prediction probability and target probability at iteration k of training respectively. $\langle \cdot, \cdot \rangle$ is the inner product function, $\mathcal{S}(\cdot)$ is the softmax function. The target can be set as:

$$\begin{aligned} \boldsymbol{t}_i(k) =& \omega\left(\gamma^k \mathcal{S}(\boldsymbol{C}\boldsymbol{p}_i(k)) + (1 - \gamma^k)\mathcal{S}(\boldsymbol{p}_i(k))\right) \\ &+ (1 - \omega)\boldsymbol{t}_i(k-1), \end{aligned} \tag{13}$$

where $\boldsymbol{C} \in \mathbb{R}^{|\mathcal{T}|\times|\mathcal{T}|}$ is the type co-occurrence matrix, $0 < \omega < 1$ is the momentum, $0 < \gamma < 1$ is the multiplication factor. For those negative types ($(e,t) \notin \mathcal{G}$) with high confidence to be positive, we directly correct them to positive (Li et al., 2021).

4.4 Optimization

For training, we adopt false-negative aware (FNA) loss function (Pan et al., 2021):

$$\begin{aligned}\mathcal{L}_{FNA} = &- \sum_{(e_i,t_j)\notin\mathcal{G}} \mu_1 \left(p_i^j - (p_i^j)^2\right) \log\left(1 - p_i^j\right) \\ &- \sum_{(e_i,t_j)\in\mathcal{G}} \log p_i^j,\end{aligned} \tag{14}$$

where p_i^j denotes the prediction probability of the i-th entity's j-th type, μ_1 is a hyper-parameter used to control the overall weight of negative samples. The FNA loss function will assign lower weight to those negative examples with too large or too small relevance scores. By combining $\mathcal{L}_{FNA}$ and $\mathcal{L}_{TCR}$, we can get the final optimization goal:

$$\mathcal{L} = \mathcal{L}_{FNA} + \mu_2 \mathcal{R}_{TCR}, \tag{15}$$

where μ_2 is the hyper-parameter.

5 Experiments

5.1 Datasets and Evaluation Metrics

Datasets. We evaluate our proposed method on two real-world KGs, including FB15k (Bordes et al., 2013) and YAGO43k (Moon et al., 2017) which are subsets of Freebase (Bollacker et al., 2008) and YAGO (Suchanek et al., 2007), respectively. Two entity typing datasets FB15kET and YAGO43kET (Moon et al., 2017) provide entity type instances by mapping entities from FB15k and YAGO43k into their entity types. The statistics of the two datasets are shown in the Appendix A.

Evaluation Metrics. For each test sample, we first calculate the relevance score between the entity and every type. Then, we rank these scores in ascending order. For a fair comparison with previous work (Zhao et al., 2020), we also adopt the filtered setting (Bordes et al., 2013) to remove all the known types in the training, validation, and test sets, before calculating score ranking. Following state-of-the-art baselines (Zhao et al., 2020; Pan et al., 2021; Zhuo et al., 2022), we adopt Mean Rank (MR), Mean Reciprocal Rank (MRR) and Hits@{1,3,10} as evaluation metrics.

5.2 Implementation Details

Our implementation is based on DGL[3] and Pytorch[4]. We use the Adam algorithm (Kingma and Ba, 2015) to optimize model parameters. The learning rate is initialized as 1e-3. The embedding dimension is set to 100, the same as previous methods to ensure fairness. All experiments are conducted with NVIDIA GeForce RTX 3090 GPUs. We select the best model leading to the highest MRR on the validation set. The best-performance hyper-parameter settings are listed in the Appendix B.

5.3 Baselines

We compare our approach MiNer with previous state-of-the-art methods, which can be divided into two categories:

Embedding-based methods: Firstly, we compare our method with classical knowledge graph embedding methods, including TransE (Bordes et al., 2013), ComplEx (Trouillon et al., 2016) and RotatE (Sun et al., 2019). Then we compare with two methods proposed specifically for the KGET task, including ETE (Moon et al., 2017) and ConnectE (Zhao et al., 2020).

Graph neural network-based methods: We also compare our method with more competitive graph neural network-based methods, including R-GCN (Schlichtkrull et al., 2018), CET (Pan et al., 2021), AttEt (Zhuo et al., 2022) and ConnectE-MRGAT (Zhao et al., 2022).

5.4 Overall Results

The performance of all the methods on the FB15kET and YAGO43kET datasets is shown in Table 1. We note the following key observations throughout our experiments:

(1) Our method outperforms all the baselines by a large margin on the two datasets. For example, compared with the state-of-the-art model CET (Pan et al., 2021), our method MiNer achieves 3.1% and 1.8% improvements of MRR on the FB15kET and YAGO43kET, respectively. It indicates that our proposed method is very effective for this task.

(2) Compared with the embedding-based methods, graph neural network-based methods achieve better performance. This suggests that the neighbor information is important for the task. However, most graph neural network-based methods only uti-

[3] https://github.com/dmlc/dgl/
[4] https://pytorch.org/

Model	FB15kET					YAGO43kET				
	MRR	**MR**	**Hit@1**	**Hit@3**	**Hit@10**	**MRR**	**MR**	**Hit@1**	**Hit@3**	**Hit@10**
Embedding-based Methods										
TransE	0.618	18	0.504	0.686	0.835	0.427	393	0.304	0.497	0.663
ComplEx	0.595	20	0.463	0.680	0.841	0.435	631	0.316	0.504	0.658
RotatE	0.632	18	0.523	0.699	0.840	0.462	316	0.339	0.537	0.695
ETE	0.500	-	0.385	0.553	0.719	0.230	-	0.137	0.263	0.422
ConnectE	0.590	-	0.496	0.643	0.799	0.280	-	0.160	0.309	0.479
Graph Neural Network-based Methods										
R-GCN ($h = 1$)	0.679	20	0.597	0.722	0.843	0.372	397	0.281	0.409	0.549
R-GCN ($h = 2$)	0.664	29	0.580	0.709	0.830	0.360	587	0.273	0.392	0.532
MRGAT ($h = 2$)	0.630	-	0.562	0.663	0.804	0.320	-	0.243	0.343	0.482
AttEt ($h = 1$)	0.620	-	0.517	0.677	0.821	0.350	-	0.244	0.413	0.565
CET ($h = 1$)	0.697	19	0.613	0.745	0.856	0.503	250	0.398	0.567	0.696
Our Method										
MiNer	**0.728**	**15**	**0.654**	**0.768**	**0.875**	**0.521**	**223**	**0.412**	**0.589**	**0.714**

Table 1: Experimental results on the FB15kET and YAGO43kET datasets. Bold denotes best results. The results of the baselines are taken from corresponding original papers. h denotes the number of hops.

Setting		FB15kET					YAGO43kET				
		MRR	**MR**	**Hit@1**	**Hit@3**	**Hit@10**	**MRR**	**MR**	**Hit@1**	**Hit@3**	**Hit@10**
Baseline (w/o Neighbor Information)											
RotatE		0.632	18	0.523	0.699	0.840	0.462	316	0.339	0.537	0.695
Our Method											
One-hop Neighbor		0.716	18	0.637	0.761	0.865	0.512	245	0.402	0.580	0.710
Multi-hop Nieghbor	$h = 2$	0.724	15	0.647	0.766	0.873	0.499	285	0.387	0.572	0.701
	$h = 3$	0.721	17	0.644	0.764	0.873	0.499	266	0.386	0.571	0.702
	$h = 4$	0.726	16	0.652	0.766	0.871	0.502	272	0.390	0.573	0.701
One-hop Neighbor + Mult-hop Nieghbor	$h = 2$	0.726	15	0.653	0.764	0.871	**0.521**	**223**	**0.412**	**0.589**	**0.714**
	$h = 3$	0.726	16	0.651	0.766	0.872	0.520	224	0.411	**0.589**	0.713
	$h = 4$	**0.728**	**15**	**0.654**	**0.768**	**0.875**	0.518	245	0.409	0.587	0.711

Table 2: Experimental results by using different neighbors on the FB15kET and YAGO43kET datasets.

lize one-hop neighbor information, ignoring multi-hop neighbor information.

(3) Traditional graph neural networks can aggregate multi-hop neighbor information. However, two-layer R-GCN performs worse than the one-layer R-GCN. The reason may be that simple information aggregations introduce a lot of noise. By contrast, our method can effectively mitigate the impact of irrelevant information.

5.5 Effectiveness of Neighbor Information Aggregation

We validate the effectiveness of neighbor information aggregation module from both one-hop neighbors and multi-hop neighbors. The results are shown in Table 2, we can observe that:

(1) Both one-hop and multi-hop neighbor information contribute to inferring the missing types. The performance improvement of using the multi-hop neighbors is more evident than that of using the one-hop neighbors. We guess that the multi-hop neighbors can provide more clues for inference. Moreover, simultaneously using these two kinds of neighbors can further improve the performance.

(2) For the FB15kET dataset, the performance is best when the hops number $h = 4$, while the hops number $h = 2$ is enough for the YAGO43kET dataset. This phenomenon may be attributed to the fact that the graph of FB15kET is more sparse than the graph of YAGO43kET. In fact, our MiNer can work under multiple hops numbers, but too many hops will lead to the over-smoothing problem.

5.6 Effectiveness of Entity Type Inference

We verify the effectiveness of entity type inference module from both type-specific local inference and type-agnostic global inference. The results are shown in Table 3. We have two important

Setting		FB15kET					YAGO43kET				
		MRR	MR	Hit@1	Hit@3	Hit@10	MRR	MR	Hit@1	Hit@3	Hit@10
Baseline (w/o Type Inference)											
R-GCN		0.679	20	0.597	0.722	0.843	0.372	397	0.281	0.409	0.549
Our Method											
Local		0.685	19	0.606	0.724	0.843	0.504	265	0.392	0.575	0.704
Global		0.684	19	0.603	0.726	0.845	0.399	319	0.302	0.442	0.584
Local	$H=2$	0.727	15	0.652	0.767	0.873	0.516	**211**	0.407	0.586	0.711
	$H=3$	0.727	15	0.653	0.768	0.874	0.516	232	0.408	0.582	0.710
+	$H=4$	0.727	15	0.652	**0.769**	0.874	0.512	231	0.403	0.581	0.706
	$H=5$	**0.728**	**15**	**0.654**	0.768	**0.875**	**0.521**	223	**0.412**	**0.589**	**0.714**
Global	$H=6$	0.726	15	0.652	0.767	0.873	0.516	221	0.408	0.584	0.712

Table 3: Experimental results by using different inference methods on the FB15kET and YAGO43kET datasets. "Local" and "Global" refer to "type-specific local inference" and "type-agnostic global inference". H denotes the number of heads.

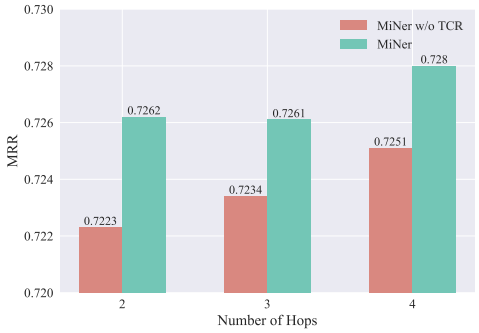

Figure 3: MRR scores for different number of hops settings on the FB15kET dataset.

Type	Golden	One-hop	Multi-hop	One+Multi-hop
/award_winner	0	0.693	0.033	0.106
/legal/topic	0	0.283	0.538	0.457
/film/actor	0	0.067	0.003	0.007
/athletics/topic	1	0.995	0.393	0.780
/naval_combatant	1	0.381	0.698	0.608
/fictional_setting	1	0.752	0.691	0.710

Table 4: Prediction probabilities of the entity /m/0f8l9c (*France*) for some types. "1" or "0" indicates the entity is with or without this type.

observations:

(1) For the FB15kET dataset, type-specific local inference and type-agnostic global inference work equally well. For the YAGO43kET dataset, type-specific local inference performs better than type-agnostic global inference. This empirically confirms that type-specific local inference works well with more entity types.

(2) Simultaneously using these two kinds of inference can further improve performance. Meanwhile, the multi-head attention mechanism plays an important role, especially when the number of attention heads $H = 5$.

5.7 Effectiveness of Type Co-occurrence Regularization

We validate the effectiveness of type co-occurrence regularization (TCR) module. The results are shown in Figure 3. Overall, we can observe that:

(1) TCR can further improve the performance of our method. This is because TCR can alleviate the memorization of the false-negative types.

(2) TCR works well under the different number of hops settings, which indicates that the module is not sensitive to the number of hops.

5.8 Case Study

We conduct a case study to verify the effectiveness of our method. Table 4 shows some prediction results for the entity /m/0f8l9c, which refers to *France*. We can observe that one-hop neighbors and multi-hop neighbors are both critical. Take /award_winner type as an example, only using one-hop neighbor information makes the wrong inference (i.e., it predicts that /m/0f8l9c has this type with higher probability), while our method can make the correct inference based on the multi-hop neighbor information. It proves that the multi-hop neighbor information is essential for the task.

6 Conclusion

In this paper, we propose a novel method called MiNer to mine treasured neighbors. First, MiNer aggregates one-hop neighbor and multi-hop neighbor information. Then, MiNer predicts the entity types by type-specific local inference and type-agnostic global inference. Finally, we use type co-occurrence regularization to prevent our model from overfitting the false-negative samples. Experi-

mental results on two widely used datasets indicate that our approach significantly outperforms previous state-of-the-art methods.

Limitations

Although our approach has worked well, there are still some limitations to be resolved in the future. The primary limitation is how to perform more efficient inference on the KGs? Our method needs to aggregate all the candidate neighbors, then mine the treasured neighbors for inferring entity types. We call this kind of method **Post-mining**. **Post-mining** methods will introduce some unrelated information when aggregating neighbors. However, **Pre-mining** methods can select valuable neighbors during the aggregation stage. **Pre-mining** methods are computationally efficient, but designing a reasonable criteria to choose neighbors is non-trivial. We will investigate it in the future work.

Acknowledgements

We thank the anonymous reviewers for their constructive comments. This work is supported by the National Key Research and Development Program of China (No. 2020AAA0106400), the National Natural Science Foundation of China (No. 62176257, 61976211, 61922085). This work is also supported by the Strategic Priority Research Program of Chinese Academy of Sciences (Grant No. XDA27020200), the Youth Innovation Promotion Association CAS, and Yunnan Provincial Major Science and Technology Special Plan Projects (No.202103AA080015).

References

Devansh Arpit, Stanisław Jastrzębski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, et al. 2017. A closer look at memorization in deep networks. In *Proceedings of the 34th International Conference on Machine Learning*.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*.

Antoine Bordes, Nicolas Usunier, Alberto Garcia-Duran, Jason Weston, and Oksana Yakhnenko. 2013. Translating embeddings for modeling multi-relational data. In *Proceedings of Advances in Neural Information Processing Systems*.

Linlin Chao, Jianshan He, Taifeng Wang, and Wei Chu. 2021. PairRE: Knowledge graph embeddings via paired relation vectors. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*.

Shuang Chen, Jinpeng Wang, Feng Jiang, and Chin-Yew Lin. 2018. Improving entity linking by modeling latent entity type information. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Xiangyu Dong, Wenhao Yu, Chenguang Zhu, and Meng Jiang. 2021. Injecting entity types into entity-guided text generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*.

Hailong Jin, Lei Hou, Juanzi Li, and Tiansi Dong. 2018. Attributed and predictive entity embedding for fine-grained entity typing in knowledge bases. In *Proceedings of the 27th International Conference on Computational Linguistics*.

Hailong Jin, Lei Hou, Juanzi Li, and Tiansi Dong. 2019. Fine-grained entity typing via hierarchical multi graph convolutional networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proceedings of the 3rd International Conference on Learning Representations*.

Thomas N. Kipf and Max Welling. 2017. Semi-supervised classification with graph convolutional networks. In *Proceedings of International Conference on Learning Representations*.

Changchun Li, Ximing Li, Lei Feng, and Jihong Ouyang. 2021. Who is your right mixup partner in positive and unlabeled learning. In *Proceedings of International Conference on Learning Representations*.

Sheng Liu, Jonathan Niles-Weed, Narges Razavian, and Carlos Fernandez-Granda. 2020. Early-learning regularization prevents memorization of noisy labels. In *Proceedings of Advances in Neural Information Processing Systems*.

Changsung Moon, Paul Jones, and Nagiza F Samatova. 2017. Learning entity type embeddings for knowledge graph completion. In *Proceedings of the 2017 ACM on conference on information and knowledge management*.

Arvind Neelakantan and Ming-Wei Chang. 2015. Inferring missing entity type instances for knowledge base completion: New dataset and methods. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

Weiran Pan, Wei Wei, and Xian-Ling Mao. 2021. Context-aware entity typing in knowledge graphs. In *Findings of the Association for Computational Linguistics: EMNLP 2021*.

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

Chao Shang, Yun Tang, Jing Huang, Jinbo Bi, Xiaodong He, and Bowen Zhou. 2019. End-to-end structure-aware convolutional networks for knowledge base completion. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*.

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

Théo Trouillon, Johannes Welbl, Sebastian Riedel, Éric Gaussier, and Guillaume Bouchard. 2016. Complex embeddings for simple link prediction. In *Proceedings of The 33rd International Conference on Machine Learning*.

Shikhar Vashishth, Rishabh Joshi, Sai Suman Prayaga, Chiranjib Bhattacharyya, and Partha Talukdar. 2018. Reside: Improving distantly-supervised neural relation extraction using side information. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*.

Shikhar Vashishth, Soumya Sanyal, Vikram Nitin, and Partha Talukdar. 2020. Composition-based multi-relational graph convolutional networks. In *International Conference on Learning Representations*.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph attention networks. In *International Conference on Learning Representations*.

Zhen Wang, Jianwen Zhang, Jianlin Feng, and Zheng Chen. 2014. Knowledge graph embedding by translating on hyperplanes. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Ruobing Xie, Zhiyuan Liu, and Maosong Sun. 2016. Representation learning of knowledge graphs with hierarchical types. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence (IJCAI-16)*.

Yu Zhao, Anxiang Zhang, Ruobing Xie, Kang Liu, and Xiaojie Wang. 2020. Connecting embeddings for knowledge graph entity typing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*.

Yu Zhao, Han Zhou, Anxiang Zhang, Ruobing Xie, Qing Li, and Fuzhen Zhuang. 2022. Connecting embeddings based on multiplex relational graph attention networks for knowledge graph entity typing. *IEEE Transactions on Knowledge and Data Engineering*.

Ke Zhu and Jianxin Wu. 2021. Residual attention: A simple but effective method for multi-label recognition. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*.

Jianhuan Zhuo, Qiannan Zhu, Yinliang Yue, Yuhong Zhao, and Weisi Han. 2022. A neighborhood-attention fine-grained entity typing for knowledge graph completion. In *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*.

A Statistics of Datasets

The statistics of the FB15kET and YAGO43kET datasets are shown in the Table 5.

Statistics	FB15kET	YAGO43kET
#Entity ($\|\mathcal{E}\|$)	14,951	42,334
#Relation ($\|\mathcal{R}\|$)	1,345	37
#Type ($\|\mathcal{T}\|$)	3,584	45,182
#Tuple ($\|\mathcal{G}\|$)	483,142	331,686
#Train	136,618	375,853
#Valid	15,848	43,111
#Test	15,847	43,119

Table 5: Statistics of FB15kET and YAGO43kET.

B Hyper-parameter Settings

Parameters	FB15kET Settings	YAGO43kET Settings
α	{0.2, **0.3**, 0.4, 0.5}	{0.6, 0.7, 0.8, **0.9**}
β_1	{0.5, **1.0**, 1.5}	{0.5, 1.0, **1.5**}
β_2	{0.5, **1.0**, 1.5}	{0.5, **1.0**, 1.5}
λ	{**0.3**, 0.6, 0.9}	{0.3, **0.6**, 0.9}
h	{2, 3, **4**}	{**2**, 3, 4}
H	{2, 3, 4, **5**, 6}	{2, 3, 4, **5**, 6}
γ	{0.3, **0.5**, 0.7}	{0.3, **0.5**, 0.7}
ω	{0.5, **0.7**, 0.9}	{0.5, **0.7**, 0.9}
μ_2	{1, 2, **3**}	{**1**, 2, 3}

Table 6: The hyper-parameter settings of the FB15kET and YAGO43kET datasets.

As shown in Table 6, α denotes the weight of skip connection, β_1 denotes the weight of type-specific local score, β_2 denotes the weight of type-agnostic global score, λ denotes the weight of one-hop neighbors, h denotes the number of hops, H denotes the number of heads, γ denotes the momentum, ω denotes the multiplicition factor and μ_2 denotes the weight of regularization.