Learning Adaptive Propagation for Knowledge Graph Reasoning

Yongqi Zhang\textsuperscript{1} Zhanke Zhou\textsuperscript{2} Quanming Yao\textsuperscript{3} Xiaowen Chu\textsuperscript{4} Bo Han\textsuperscript{2}
\textsuperscript{1}4Paradigm Inc., Beijing, China
\textsuperscript{2}Department of Computer Science, Hong Kong Baptist University, Hong Kong, China
\textsuperscript{3}Department of Electronic Engineering, Tsinghua University, Beijing, China
\textsuperscript{4}Data Science and Analytics Thrust, HKUST (Guangzhou), Guangzhou, China
zhangyongqi@4paradigm.com, {cszkkzhou,bhanml}@comp.hkbu.edu.hk
qyaoaa@tsinghua.edu.cn, xwchu@ust.hk

Abstract

Due to the success of Graph Neural Networks (GNNs) in learning from graph-structured data, various GNN-based methods have been introduced to learn from knowledge graphs (KGs). In this paper, to reveal the key factors underneath existing GNN-based methods, we revisit exemplar works from the lens of the propagation path. We find that the answer entity can be close to queried one, but the information dependency can be long. Thus, better reasoning performance can be obtained by exploring longer propagation paths. However, identifying such a long-range dependency in KG is hard since the number of involved entities grows exponentially. This motivates us to learn an adaptive propagation path that filters out irrelevant entities while preserving promising targets during the propagation. First, we design an incremental sampling mechanism where the close and promising target can be preserved. Second, we design a learning-based sampling distribution to identify the targets with fewer involved entities. In this way, GNN can go deeper to capture long-range information. Extensive experiments show that our method is efficient and achieves state-of-the-art performances in both transductive and inductive reasoning settings, benefiting from the deeper propagation.

1 Introduction

A knowledge graph (KG) is a graph-structural data with many relational facts \cite{1,16}. The facts can be formed as triplets \textit{(subject entity, relation, object entity)}, denoted as \((e_s, r, e_o)\). KG reasoning is a task that deduces new facts from existing ones \cite{6}. The atomic reasoning problem is denoted as a query \((e_q, r_q, ?)\), with the query entity \(e_q\) and query relation \(r_q\). Our goal is to find the target answer entity \(e_a\) for the specific query. KG reasoning is attracting growing interest, and it has been widely applied in recommendation \cite{4,32}, question answering \cite{14} and drug interaction prediction \cite{39}.

Given the query \((e_q, r_q, ?)\), a KG reasoning model is expected to use pieces of evidence from the local neighborhoods to correctly find the target answer entity \(e_a\). Pioneer triplet-based models, such as TransE \cite{3}, ConvE \cite{9} and QuatE \cite{42}, directly score each candidate answer entity through the learned entity and relation embeddings to implicitly capture the local evidence. The path-based methods, like MINERVA \cite{8}, DRUM \cite{25} and RNNLogic \cite{23}, learn the relational paths starting from \(e_q\) and explore candidate entities that are more likely to be the target answer entity \(e_a\). Although these models well capture the semantic patterns in KGs, they fail to capture the more complex topological patterns there.

\textsuperscript{1}This work was performed when Z. Zhou was an intern in 4Paradigm. Z. Zhang and Z. Zhou have equal contribution, and correspondence is to Q. Yao

Preprint. Under review.
As discussed, previous works focus on the manual design of propagation schemes. There lacks a deep analysis of how recent improvements are gained by the propagation scheme. To fill this gap, we first revisit the existing methods in Section 2.1.1 and show the necessity to propagate deeper from the perspective of the propagation path in Section 2.1.2.

Figure 1: An example of KG reasoning and three kinds of propagation path. Full propagation propagates over the full set of entities \( V \). Progressive propagation starts from the query entity \( e_q \) and gradually propagates to its \( \ell \)-hop neighbors in the \( \ell \)-th step. The proposed adaptive propagation selects the informative entities to propagate.

Due to the success of graph neural networks (GNNs) \([1, 10, 17]\) in modeling the graph-structured data, recent works \([30, 28, 43, 44]\) attempt to utilize the power of GNNs for KG reasoning. They generally follow the message propagation framework \([10, 17]\), where the representations of entities are propagated to their neighborhoods for multiple steps. R-GCN \([26]\) and CompGCN \([30]\) propagate the low-level entity embeddings among all the neighboring entities to obtain the high-level embeddings. These methods are limited to two to four propagation steps due to the over-smoothing problem of GNNs in the full neighborhood \([21]\). GraIL \([28]\) extracts the enclosing subgraph between \( e_q \) and \( e_a \), and propagates within the subgraph. It shows that GNN can work in inductive reasoning, where there are new entities in inference. However, GraIL will be extremely expensive when too many candidate entities are expected to be scored. Recently, NBFNet \([44]\) and RED-GNN \([43]\) progressively propagate from the query entity \( e_q \) to its \( \ell \)-hop neighborhood. By differentiating the neighboring entities as well as their representations for different queries, they achieve state-of-the-art performance in both transductive and inductive settings by propagating deeper.

To reveal the key factors underneath these GNN-based methods, we revisit exemplar works from the lens of propagation path as in Fig.1. We observe that while the target answer entity can be close to the queried one, remote entities are still informative to help distinguish the target, i.e., the information dependency can be long. The better reasoning performance of progressive propagation methods, i.e., NBFNet and RED-GNN, are obtained by exploiting longer information through deeper propagation paths. However, identifying such a long-range dependency in KG is hard since the number of involved entities grows exponentially as depth grows. This motivates us to learn an adaptive propagation path named as AdaProp, which can filter out irrelevant entities while preserving the promising targets during propagation. First, we design an incremental sampling mechanism where the promising target can be preserved in the propagation steps. Second, we design a learning-based sampling method to identify the promising targets with fewer involved entities. In this way, our method can go deeper to capture the long-range information. The main contributions are summarized as follows:

- A unified framework to revisit GNN-based KG reasoning methods through the lens of propagation path, which reveals remote entities are still informative to distinguish the nearby target.
- A learning-based sampling method is proposed to identify the promising targets while filtering out the irrelevant entities during propagation, helping the GNN to propagate deeper in KG reasoning.
- The state-of-the-art performance in both transductive and inductive reasoning settings.

**Notations.** In this paper, representations and parameters are denoted by lowercase boldface, e.g., \( h, w, \theta \). Sets are denoted by script fonts, e.g., \( \mathcal{E}, \mathcal{V} \). A KG is in the form of \( K = \{ \mathcal{V}, \mathcal{R}, \mathcal{E}, \mathcal{Q} \} \), where \( \mathcal{V}, \mathcal{R} \) are the sets of entities and relations, \( \mathcal{E} = \{ (e_s, r, e_o) | e_s, e_o \in \mathcal{V}, r \in \mathcal{R} \} \) is the set of fact triples, and \( \mathcal{Q} = \{ (e_q, r_q, e_a) | e_q, e_a \in \mathcal{V}, r \in \mathcal{R} \} \) is the set of query triplets.

2 Proposed Method

2.1 Revisiting existing works through the lens of propagation path

As discussed, previous works focus on the manual design of propagation schemes. There lacks a deep analysis of how recent improvements are gained by the propagation scheme. To fill this gap, we first revisit the existing methods in Section 2.1.1 and show the necessity to propagate deeper from the perspective of the propagation path in Section 2.1.2.
To reveal the key factors underneath the GNN-based methods, we summarize the general propagation methods, such as RED-GNN \[43\] and NBFNet \[44\], start the propagation constrained propagation method GraIL \[28\] propagates within the enclosing subgraph, i.e., CompGCN \[30\], propagate within the full neighborhood, i.e., R-GCN \[26\] and NBFNet and RED-GNN, are provided in Appendix C.3. The other datasets in Appendix C.3 have the same observations. (a).left: the average values of \(\text{IE}(\hat{G}^L)\) for different \(L\); (a).right: the performance of different validation samples, grouped by the distance (hops) from \(e_q\) to \(e_a\) for CompGCN \((L=3\) and \(L=5\)), RED-GNN \((L=5\) and \(L=8\)) and the proposed method \((L=5\) and \(L=8\)). (b): The distance distribution from \(e_q\) to \(e_a\).

2.1.1 General framework

To reveal the key factors underneath the GNN-based methods, we summarize the general propagation framework for KG reasoning in Alg.1 based on the message propagation framework in GNNs \[10\] \[17\]. At the \(l\)-th step, the messages are first computed on the edges in \(\mathcal{E}^l\) based on entity representations \(h_{e_o}^l\) and relation representations \(h_{e}^l\) as in line 2 and then propagated to the entities in \(\mathcal{V}^l\) followed by an activation function \(\delta(\cdot)\) in line 3. The specified functions \(\text{MESS}(\cdot)\) and \(\text{AGG}(\cdot)\) for different methods, including R-GCN, CompGCN, GraIL, NBFNet and RED-GNN, are provided in Appendix A.1. After \(L\)-steps’ propagation, the entity representations \(h_{e_o}^L\) are obtained to measure the plausibility of each candidate entities \(e_o \in \mathcal{V}^L\). The key difference between Alg.1 and the plain GNNs \[10\] \[17\] is that the propagation path and the representations \(h_{e_o}^L\)'s here can be query-dependent.

In other words, the message propagation can be different for different query \((e_q, r, e, \cdot, ?)\).

Based on Alg.1, we are able to define the core concept of “propagation path” in Def.1 and classify the most general GNN-based methods.

**Definition 1 (The propagation path).** The propagation path for a query \((e_q, r, e, \cdot, ?)\) is composed of involved entities in each propagation step as in Algorithm 1. The involved entities in the \(l\)-th step are termed as \(\mathcal{V}^l\) \((l=0, 1, \ldots, L, \mathcal{V}^L \subseteq \mathcal{V})\). Namely, a \(L\)-layer propagation path \(\hat{G}^L = \{\mathcal{V}^0, \mathcal{V}^1, \ldots, \mathcal{V}^L\}\).

2.1.2 The need of propagating deeper

Now, it is clear that the existing GNN-based methods can be represented by \(F(w, \hat{G}^L)\) with network weights \(w\) and propagation path \(\hat{G}^L\). The full propagation methods, such as R-GCN \[26\] and CompGCN \[30\], propagate within the full neighborhood, i.e., \(|\mathcal{V}^l| = |\mathcal{V}|\) for \(l = 0 \ldots L\). The constrained propagation method GraIL \[28\] propagates within the enclosing subgraph, i.e., \(|\mathcal{V}^l| = |\mathcal{V}^l_{(e_q,e_o)}|\) for \(l = 0 \ldots L\) and \(\mathcal{V}^l_{(e_q,e_o)} \subseteq \mathcal{V}\) is the enclosing subgraph between \(e_q\) and \(e_o\). The progressive propagation methods, such as RED-GNN \[43\] and NBFNet \[44\], start the propagation from the query entity \(e_q\) and progressively propagate to the \(L\)-hop neighborhood of \(e_q\), i.e., \(\mathcal{V}^0 = \{e_q\}\) and \(\mathcal{V}^l = \cup_{e \in \mathcal{V}^{l-1}} N(e)\) for \(l = 1 \ldots L\), where \(N(e)\) denotes the set of neighboring entities of \(e\).

To quantitatively compare different designs of propagation path, we introduce two metrics below:

\[
\text{Ratio of involved entities: } \text{IE}(\hat{G}^L) = |\cup_{l=1 \ldots L} \mathcal{V}^l|/|\mathcal{V}| \tag{1}\\
\text{Ratio of target over candidates: } \text{ToC}(\hat{G}^L) = |\{e_o \in \mathcal{V}^L\}|/|\mathcal{V}^L|, \tag{2}
\]

\[
\text{Algorithm 1 General propagation framework for KG reasoning.}\\
\text{Require: query } (e_q, r, e, \cdot, ?), \text{ the propagation path } \hat{G}^L \in \{\mathcal{V}^0, \mathcal{V}^1, \ldots, \mathcal{V}^L\}, \text{ functions MESS(\cdot) and AGG(\cdot).}\\
1: \text{for } l = 1 \ldots L \text{ do}\\
2:\text{obtain } m^l_{(e_o, r, e_o)} := \text{MESS}(\hat{h}^{l-1}_{e_o}, \hat{h}^{l-1}_r, h^l_r) \text{ for all the edges } \mathcal{E}^l = \{(e_o, r, e_o) | e_o \in \mathcal{V}^{l-1}, e_o \in \mathcal{V}^l\};\\
3:\text{obtain } h^l_{e_o} := \delta(\text{AGG}(m^l_{(e_o, r, e_o)}), (e_o, r, e_o) \in \mathcal{E}^l) \text{ for all the entities } e_o \in \mathcal{V}^l;\\
4:\text{end for}\\
5:\text{return } h^L_{e_o} \text{ for each } e_o \in \mathcal{V}^L.
\]
Comparing the per-hop performance for GNNs with three propagation schemes, in terms of the two factors as well as the per-hop performance.

(i). The majority of target answer entity \( e_a \) is less than 6 steps away from the query entity \( e_q \).

(ii). Comparing full propagation and progressive propagation, the progressive propagation can propagate deeper with smaller \( \text{IE}(\hat{G}^L) \) and larger \( \text{ToC}(\hat{G}^L) \). The target answer entity in the progressive propagation is easier to be distinguished, thus leading to better performance.

(iii). Comparing the per-hop performance for GNNs with \( L = 5 \) and \( L = 8 \), even though a propagation with \( L = 5 \) can cover the targets within 5 hops, a deeper GNN with \( L = 8 \) can achieve better per-hop performance by capturing the longer-range information.

(iv). For progressive propagation, the improvement of \( L = 8 \) over \( L = 5 \) is marginal since it is harder to tell the true target apart from too many candidates. In comparison, the adaptive propagation has larger \( \text{ToC}(\hat{G}^L) \) for deeper depth and achieves the better per-hop performance.

For short, even though the target answer entities can be close to the queried one, it is important for the propagation path to go deeper such that the long-range information can be captured.

### 2.2 Learning to propagate by sampling

As in Section 2.1, a good design of propagation path should be able to reduce the number of involved entities while preserving the promising targets such that it can go deeper. However, this is challenging since there are many entities in KGs. Even though the progressive propagation can propagate deeper with fewer involved entities at the early steps, it is still not good enough when \( L \) gets larger.

The existing sampling techniques \([5, 11, 38, 41, 45]\) that are designed for homogeneous graphs do not work well here since (1) the answer entity, which is unknown before the reasoning, should be preserved in \( V^L \) in the last step; (2) the edges in KG have multiple types and the pattern across relations is complex \([16, 31]\); (3) the informative entities are distinct for different queries \([8, 37]\).

Thus, we argue that a specially designed sampling approach for KG reasoning is needed.

In this part, we propose to learn an adaptive propagation path \( \hat{G}^L(\theta) \), which is parameterized by the sampling parameters \( \theta \), for KG reasoning. The GNN \( F(w, \hat{G}^L(\theta)) \) contains both the network parameters \( w \) and the sampling parameters \( \theta \). Given the set of training queries \( Q_{tra} \), we have the following objective to optimize the parameters:

\[
\theta^*, w^* = \arg \min_{\theta, w} \mathcal{L}(F(w, \hat{G}^L(\theta)), Q_{tra}),
\]

where the loss function \( \mathcal{L} = -\sum_{e_q \sim r, e_a} \sum_{e_o \in V^L} y_{e_a} \log(\phi_{e_a}) + (1 - y_{e_a}) \log(1 - \phi_{e_a}) \) with the candidate score \( \phi_{e_a} = f(h_{e_o}, w_o) \) and candidate label \( y_{e_a} = 1 \) if \( e_o = e_a \) otherwise 0. To solve this problem, we propose an incremental sampling scheme in Section 2.2.1 and the learning-based sampling distribution in Section 2.2.2.

#### 2.2.1 Incremental sampling scheme

![Propagate with incremental sampling](image)

Figure 3: Propagate with incremental sampling. For candidate generation, \( \hat{V}^L = \text{CAND}(V^{L-1}) \) are formed by the newly-visited entities in green. After candidate sampling, the sampled entities \( \text{SAMP}(\hat{V}^L) \), i.e., the entity 6 and 7, are added to the propagation path for propagating afterwards.

\( ^3 \)Since the extraction of enclosing subgraph in the constraint propagation is very expensive and it empirically performs bad, we discuss the limitations of this type in Appendix A.2.
When the depth goes deeper, there will be more entities in the candidate set with temperature.

Alg. 2 has several advantages. First, the number of involved entities is linearly bounded by the number of entities in the candidate set. We choose the straight-through (ST) estimator [2, 15] rather than the reparameterization trick [15] without replacement from the candidate set.

The full procedure is shown in Alg. 2. Given the entities \( V \) with the highest score evaluated based on the scoring function \( f(h_{e_o}; w_o) \) is regarded as the predicted target.

Candidate generation. Given the set of entities in the \((\ell - 1)\)-th step \( \bar{V}^{\ell-1} \), we denote the candidate entities to sample in the \(\ell\)-th step as \( \mathcal{CAND}(\bar{V}^{\ell-1}) \). Basically, all the neighboring entities in \( \mathcal{V}^{\ell-1} \) will be preserved in the next sampling step, we regard the newly-visited entities as the candidates, i.e., \( \mathcal{CAND}(\bar{V}^{\ell-1}) = \mathcal{V}^{\ell-1} \). As in Fig. 3, we generate the 1-hop neighbors of \( e_o \) as candidates and directly preserve \( e_o \) in the next propagation step.

Candidate sampling. To control the number of involved entities, we sample \( K \) entities without replacement from the candidate set \( \mathcal{CAND}(\bar{V}^{\ell-1}) \). The sampled entities at \(\ell\)-th step, together with the entities in the \((\ell - 1)\)-th step, constitute the involved entities at the \(\ell\) step, i.e., \( V^\ell = \mathcal{V}^\ell \cup \text{SAMP}(\mathcal{CAND}(\bar{V}^{\ell-1})) \).

### 2.2.2 Learning-based sampling distribution

When the depth goes deeper, there will be more entities in the candidate set \( \mathcal{CAND}(\bar{V}^{\ell-1}) \). Hence, it is important to filter out the irrelevant entities and preserve the promising ones. As in Alg. 1, the entity representation \( h^{\ell}_{e_o} \) in the last step is used to measure the plausibility of \( e_o \) as the answer entity. Hence, we introduce a mapping function \( g(h^{\ell}_{e_o}; \theta) : \mathbb{R}^d \rightarrow \mathbb{R} \) with parameters \( \theta^\ell \) in each layer and sample according to the probability distribution \( p^{\ell}(e_o) := \exp \left( g(h^{\ell}_{e_o}; \theta^\ell)/\tau \right) / \sum_{e_o \in \mathcal{V}^{\ell-1}} \exp \left( g(h^{\ell}_{e_o}; \theta^\ell)/\tau \right) \) with temperature \( \tau > 0 \) by the Gumbel top-k trick [18, 34] (details in Appendix B.1).

Note that \( \mathcal{G}^{\ell}(\theta) \) is discrete. To estimate the gradient of loss function over the sampling parameters \( \theta \), we choose the straight-through (ST) estimator [2, 15] rather than the reparameterization trick [15] since we conduct explicit sampling to reduce the number of involved entities and save computation cost. An alternative way to estimate the gradient is the REINFORCE [33, 20], which has high-variance [27] and the variance will be accumulated across the propagation steps.

### 2.3 The full algorithm

The full procedure is shown in Alg. 2. Given the entities \( \mathcal{CAND}(\bar{V}^{\ell-1}) \) in the \((\ell - 1)\)-th step, we firstly obtain the neighboring entities \( \mathcal{V}^{\ell-1} \) of \( \mathcal{CAND}(\bar{V}^{\ell-1}) \) and generate the newly-visited ones \( \bar{V}^{\ell} = \mathcal{V}^{\ell-1} \setminus \mathcal{V}^{\ell-1} \) as candidates in line 3. Then, messages are computed on edges between entities in \( \mathcal{V}^{\ell-1} \) and \( \mathcal{V}^{\ell-1} \) in line 4 and aggregated to entities in \( \mathcal{CAND}(\bar{V}^{\ell-1}) \) in line 5. The top-\(K\) entities in \( \mathcal{CAND}(\bar{V}^{\ell-1}) \) are sampled according to the logits computed in line 6. Finally, the sampled entities \( \mathcal{CAND}(\bar{V}^{\ell-1}) \) are concatenated with \( \mathcal{V}^{\ell-1} \) to form the involved entities \( \mathcal{CAND}(\bar{V}^{\ell-1}) \) in the \(\ell\)-th step in line 7. After propagating for \(L\) steps, the final step representations \( h^{L}_{e_o} \) for all the entities \( e_o \in \mathcal{V}^{L} \) are returned. The candidate in \( \mathcal{V}^{L} \) with the highest score evaluated based on the scoring function \( f(h^{L}_{e_o}; w_o) \) is regarded as the predicted target.

The key difference compared with Alg. 1 is that the propagation path is no longer given before the message propagation. Instead, it is adaptively adjusted based on the entity representations and is updated in each propagation step. In addition, the hidden representations for the sampled entities go through the straight-through estimator to estimator gradient for the sampling parameters.

Alg. 2 has several advantages. First, the number of involved entities is linearly bounded by the propagation depth \( L \), i.e., \( IE(\mathcal{G}^L) \leq L \cdot K \), and \( K \ll |\mathcal{V}| \). Second, the promising target is preserved to guarantee a larger ToC(\(\mathcal{G}^L\)). Third, with the optimization problem in Eq. 3, the sampling parameters learn to filter out irrelevant entities such that the propagation path can go deeper. Finally, it also enjoys good efficiency since only a small portion of entities are involved and the extra computation cost for sampling is low. All the potential answer entities can be scored in a single forward pass, rather than propagating in the answer-specific propagation paths for multiple times like GraIL [29].
Algorithm 2 AdaProp: learning adaptive propagation path.

Require: query \((e_q, r_q, ?)\), \(\mathcal{V}^0 = \{e_q\}\), steps \(L\), \# of sampled entities \(K\), and functions \(\text{MESS}(\cdot)\) and \(\text{AGG}(\cdot)\).

1: for \(\ell = 1 \ldots L\) do
2: \hspace{1em} get the neighboring entities \(\mathcal{V}_{\text{neib}}^{\ell-1} = \bigcup_{e \in \mathcal{V}^{\ell-1}} \mathcal{N}(e)\), the newly-visited entities \(\mathcal{V}_{\text{neib}}^{\ell} = \mathcal{V}^{\ell-1} \setminus \mathcal{V}_{\text{neib}}^{\ell-1}\) and edges \(\mathcal{E}^{\ell} = \{(e_s, r, e_o) | e_s \in \mathcal{V}_{\text{neib}}^{\ell-1}, e_o \in \mathcal{V}_{\text{neib}}^{\ell} \};\)
3: \hspace{1em} obtain \(m^{\ell}_{(e_s, r, e_o)} := \text{MESS}(h^{\ell}_{e_s}, h^{\ell-1}_{e_o}, h^{\ell}_{e_o})\) for the edges \((e_s, r, e_o) \in \mathcal{E}^{\ell};\)
4: \hspace{1em} obtain \(h^{\ell}_{e_o} := \text{AGG}(m^{\ell}_{(e_s, r, e_o)} | (e_s, r, e_o) \in \mathcal{E}^{\ell})\) for the entities \(e_o \in \mathcal{V}_{\text{neib}}^{\ell};\)
5: \hspace{1em} \text{logits computation:} obtain the Gumbel logits \(G_{e_o} = g(h^{\ell}_{e_o}; \theta^\ell) - \log(-\log U_{e_o})\) with \(U_{e_o} \sim \text{Uniform}(0, 1)\) for the entities \(e_o \in \mathcal{V}^{\ell};\)
6: \hspace{1em} \text{candidate sampling:} obtain sampled entities \(\tilde{\mathcal{V}}^{\ell} = \{\arg \top_{K} G_{e_o}, \text{for } e_o \in \mathcal{V}^{\ell}\};\)
7: \hspace{1em} \text{update propagation path:} update \(\mathcal{V}^{\ell} = \mathcal{V}^{\ell-1} \cup \tilde{\mathcal{V}}^{\ell};\)
8: \hspace{1em} end for
9: \hspace{1em} return \(h^{\ell}_{e_o}\) for each \(e_o \in \mathcal{V}^{\ell};\)

3 Experiments

In this section, we empirically show the effectiveness and analyze the design components of the proposed AdaProp. All the experiments are implemented in Python with the PyTorch framework [22] and run on a single NVIDIA RTX 3090 GPU with 24GB memory.

3.1 Compare with general KG reasoning methods

We compare AdaProp with general KG reasoning methods in both transductive and inductive reasoning. In the transductive setting, the training and test set share the same set of entities and relations, i.e., all the entities and relations in the test set are visited during training. In the inductive setting, the training and test set share the same set of relations but different entities, namely the entities in the test set are not seen during training. We follow [3, 28, 31, 44] to use the filtered ranking-based metrics, i.e., mean reciprocal ranking (MRR) and Hit@k for evaluation. For both metrics, the larger value indicates the better performance. For AdaProp, we tune the number of propagation steps \(L\) from 5 to 8, the number of sampled entities \(K\) in \(\{100, 500, 1000, 2000\}\), and the sampling temperature \(\tau\) in \(\{0.5, 1.0, 2.0\}\). The details of other hyper-parameters are listed in Appendix C.2.

3.1.1 Transductive setting

Datasets. We use three widely used KG completion datasets, including WN18RR [9], FB15k237 [29] and NELL-995 [35], with tens of thousand of entities. The detailed statistics are in Appendix C.1.

Baselines. We take ConvE [9] and QuatE [42] to represent the triplet-based methods, and MINERVA [8], DRUM [25] and RNNLogic [23] to represent the path-based methods. For the GNN-based methods, we take CompGCN [30] as the full propagation method since it is better than R-GCN [26]. GraIL [28] is intractable to work on the transductive KGs with many entities. For progressive propagation methods, we compare with both RED-GNN [43] and NBFNet* [44] here.

Table 1: Transductive setting. Best performance is indicated by the bold face numbers, and the underline means the second best. “–” means unavailable results.

| type | models   | WN18RR       | FB15k237       | NELL-995       |
|------|----------|--------------|----------------|----------------|
|      |          | MRR Hit@1    | MRR Hit@10     | MRR H@1        | MRR H@10        |
| non-GNN | ConvE    | 0.43 39      | 0.325 23.7     | –              | –              |
|       | QuatE    | 0.480 44.0   | 0.350 25.6     | 0.533 46.6     | 64.3           |
|       | MINERVA  | 0.448 41.3   | 0.293 21.7     | 0.513 41.3     | 63.7           |
|       | DRUM     | 0.486 42.5   | 0.343 25.5     | –              | –              |
|       | RNNLogic | 0.483 44.6   | 0.334 25.2     | –              | –              |
| GNNs  | CompGCN  | 0.479 44.3   | 0.355 26.4     | –              | –              |
|       | RED-GNN  | 0.533 48.5   | 0.374 28.3     | 0.543 47.6     | 65.1           |
|       | NBFNet*  | 0.551 49.7   | 0.415 32.1     | 0.513 43.1     | 64.7           |
|       | AdaProp  | 0.562 50.1   | 0.497 30.9     | 0.564 50.1     | 66.7           |
Results. The comparison of AdaProp with the transductive reasoning methods is in Tab.1. First, the GNN-based methods generally perform better than the non-GNN based ones by capturing both the semantic and structural information in KGs. Second, the progressive propagating methods, i.e., RED-GNN and NBFNet, perform better than the full propagation method CompGCN since the target answer entity can be easier distinguished from the other candidates in the progressive propagation steps. Besides, CompGCN will run out-of-memory on the larger data NELL-995 as it propagates among the full KG. In comparison, AdaProp achieves leading performance in WN18RR and NELL-995, and it is the runner up in FB15k237. The hyper-parameter configurations are provided in Appendix C.2 showing that AdaProp learns deeper propagation path than the other GNN-based reasoning methods.

3.1.2 Inductive setting

Setup. Following [28, 43], we use the same subsets (four versions each and 12 subsets in total) of WN18RR, FB15k237 and NELL-995, where each subset has a different split of training set and testing set. The dataset details are in Appendix C.1. We follow [43] to evaluate the ranking of target answer entity over all the negative entities rather than 50 randomly sampled negative ones in [28].

Baselines. All the reasoning methods which learn entity embeddings in training cannot work in this setting. Hence, for path-based methods, we compare with those that learn sequential rules, i.e., RuleN [19], NeuralLP [37] and DRUM [25]. For GNN-based methods, we take GraIL, RED-GNN [43] and NBFNet [44] as baselines.

Table 2: Inductive setting (evaluated with Hit@10). Results of the other two metrics, i.e., MRR and Hit@1, are in Appendix C.4.

| type | methods | WN18RR | FB15k237 | NELL-995 |
|------|---------|--------|----------|----------|
|      |         | V1     | V2       | V3       | V4       | V1     | V2       | V3       | V4       |
| non-GNN | RuleN   | 73.0   | 69.4     | 40.7     | 68.1     | 44.6   | 59.9     | 60.0     | 60.5     | 76.0   | 51.4     | 53.1     | 48.4     |
|        | Neural LP | 77.2   | 74.9     | 47.6     | 70.6     | 46.8   | 58.6     | 57.1     | 59.3     | 87.1   | 56.4     | 57.6     | 53.9     |
|        | DRUM     | 77.7   | 74.7     | 47.7     | 70.2     | 47.4   | 59.5     | 57.1     | 59.3     | 87.3   | 54.0     | 57.7     | 53.1     |
| GNNs  | GraIL    | 76.0   | 77.6     | 40.9     | 68.7     | 42.9   | 42.4     | 42.4     | 38.9     | 56.5   | 49.6     | 51.8     | 50.6     |
|        | RED-GNN  | 79.9   | 78.0     | 52.4     | 72.1     | 48.3   | 62.9     | 60.3     | 62.1     | 86.6   | 60.1     | 59.4     | 55.6     |
|        | NBFNet   | 82.7   | 79.9     | 56.3     | 70.2     | 50.3   | 63.9     | 54.7     | 54.3     | 56.5   | 61.9     | 61.4     | 59.1     |
| AdaProp |          | 86.6   | 83.6     | 62.6     | 75.5     | 55.1   | 65.9     | 63.7     | 63.8     | 88.6   | 65.2     | 61.8     | 60.7     |

Results. The results of the Hit@10 metric are summarized in Tab.2. In the inductive setting, it is more important to learn the general relational patterns that can be generalized from the training KG to the inference KG. AdaProp performs the best in the different datasets and different split versions. This demonstrates that the long-range information is common in different KGs and it can be generalized to a new KG whose entities are not seen during training.

3.2 Understanding the proposed sampling mechanism

In this part, we evaluate the sampling mechanism from the perspectives of the sampling strategy, the importance of learning, and the propagation steps. The sampling strategies for homogeneous graph can be classified into three categories: node-wise sampling [11, 38] that samples $K$ neighbors from $N(e)$ for each entity $e$, layer-wise sampling [5, 13, 45] that samples at most $K$ entities in each layer, and subgraph sampling [7, 41, 40] that samples the local subgraph around the queried entity. In Tab.3, we especially adapt each type of method for the KG reasoning task here (design details in Appendix B.3). For node-wise sampling and layer-wise sampling, we also design a learned version with the same learning distribution in Section 2.2.2, but the subgraph sampling is hard to be learned.

We compare the different variants on a transductive dataset WN18RR with MRR metric and an inductive dataset FB15k237-v1 with Hit@10 metric. We also show the values of $\text{IE}(\hat{G}^L)$ and $\text{ToC}(\hat{G}^L)$. For fair comparison, we guarantee that the different variants have the same number of propagation steps $L$ and the similar amount of involved entities $\text{IE}(\hat{G}^L)$.

Sampling strategy. As in Tab.3, the incremental sampling is better than the other sampling strategies in both the learned and not learned setting. The success dues to the larger $\text{ToC}(\hat{G}^L)$ by preserving the previously sampled entities. However, the node-wise and layer-wise sampling perform bad since the
sampling is uncontrolled. Since the target entities are close to the queries one, the subgraph sampling is relatively better than the node-wise and layer-wise sampling by propagating within the local area around the queried entity.

**Importance of learning.**  By comparing the top and bottom part in Tab 3 we observe that learning is important for all the sampling strategies. Since the entity representations can indicate the plausibility of different entities, a parameterized sampling distribution in proportion to $g(h^L_{e_o}, \theta)$ helps to identify the promising targets, increasing ToC($\hat{G}^L$) and leading to better performance. We further provide a comparison of the straight-through estimator and REINFORCE estimator in Appendix B. Overall, the incremental sampling equipped with the learned distribution works the best.

| learn methods | WN18RR  | FB15k237-v1 |
|---------------|---------|-------------|
|               | IE($\hat{G}^L$) | ToC($\hat{G}^L$) | MRR | IE($\hat{G}^L$) | ToC($\hat{G}^L$) | Hit@10 (%) |
| not learned   |         |             |     |                  |             |           |
| Node-wise     | $1.18 \times 10^{-1}$ | $1.38 \times 10^{-4}$ | 0.453 | $3.79 \times 10^{-1}$ | $1.35 \times 10^{-3}$ | 47.9 |
| Layer-wise    | $1.23 \times 10^{-1}$ | $1.46 \times 10^{-4}$ | 0.477 | $3.48 \times 10^{-1}$ | $1.45 \times 10^{-3}$ | 46.2 |
| Subgraph      | $1.27 \times 10^{-1}$ | $1.57 \times 10^{-4}$ | 0.502 | $3.63 \times 10^{-1}$ | $1.50 \times 10^{-3}$ | 50.5 |
| Incremental   | $1.21 \times 10^{-1}$ | $1.61 \times 10^{-4}$ | 0.508 | $3.51 \times 10^{-1}$ | $1.52 \times 10^{-3}$ | 50.1 |
| learned       |         |             |     |                  |             |           |
| Node-wise     | $1.20 \times 10^{-1}$ | $1.52 \times 10^{-4}$ | 0.529 | $3.52 \times 10^{-1}$ | $1.47 \times 10^{-3}$ | 50.4 |
| Layer-wise    | $1.19 \times 10^{-1}$ | $1.64 \times 10^{-4}$ | 0.533 | $3.43 \times 10^{-1}$ | $1.55 \times 10^{-3}$ | 52.4 |
| Incremental   | $1.16 \times 10^{-1}$ | $1.78 \times 10^{-4}$ | 0.562 | $3.48 \times 10^{-1}$ | $1.57 \times 10^{-3}$ | 55.1 |

**Propagation steps.** To demonstrate that AdaProp can be benefited by learning deeper propagation steps, we compare the performance of different GNN-based reasoning methods with different propagation depth in Fig 4. As shown, CompGCN only performs well when $L \leq 4$ since the ToC($\hat{G}^L$) is low as in Fig 2(a) and it will easily suffer from the problem of over-smoothing [21]. RED-GNN and NBFNet have increasing performance from $L = 2$ to $L = 6$ by capturing the more long-range information. But the improvement is marginal for $L > 6$ when too entities are involved. As for AdaProp, the performance consistently gets improved with deeper propagation steps.

![Figure 4: Performance of different L.](image1)

![Figure 5: Learning curves.](image2)

In addition, we show the learning curves of the different GNN-based methods in Fig 5. It is clear that AdaProp is the most efficient since it propagates with fewer entities and the sampling cost is not high.

### 3.3 Case study: query-dependent propagation path

In Section 2.2.2, the sampling distribution is related to the query-dependent representations $h^\ell_{e_o}$. Here, we show that the adaptive propagation path is also query-dependent. In Fig 6, we plot the exemplar propagation paths of AdaProp and RED-GNN for two different queries $q_1 = (e_q, r_1, ?)$ and $q_2 = (e_q, r_2, ?)$, which have the same query entity but different query relation on FB15k237-v1. As shown, the propagation paths are different for AdaProp, but the same for RED-GNN. Besides, the propagation paths for AdaProp have much fewer entities.

Furthermore, we provide a quantitative evaluation in Tab 4. Given the set of query pairs $(q_1, q_2)$ that share the same query entity but different query relations. Denote $V^\ell_{q_1}$ and $V^\ell_{q_2}$ as the set of entities in the propagation path of $q_1$ and $q_2$, respectively, we use the average value of $O(q_1, q_2) = \frac{\sum_{e \in V^\ell_{q_1}} |V^\ell_{q_1} \cap V^\ell_{q_2}|}{\sum_{e \in V^\ell_{q_1}} |V^\ell_{q_1}| \cdot |V^\ell_{q_2}|}$ to indicate the overlapping of their propagation paths. As in Tab 4, the intersection is larger for larger $K$, but consistently smaller than 1. On FB15k237-v1 which has more relations than WN18RR, the intersection is lower. These results also show that AdaProp learns query-dependent propagation path.
4 Related Works

**Graph neural networks (GNNs).** The basic framework of GNNs is called message propagation \[17\] \[10\] \[36\], where the node representations are propagated for \(L\) steps among the neighborhood. The general propagation framework in Alg.1 is different with the message propagation in GNN since the propagation path \(\hat{G}^L\) and entity representations \(h^l_e, l = 1 \ldots L\) can be different for different queries.

**Sampling methods for GNNs.** The classical message propagation is hard to scale to large graphs \[12\]. Several sampling methods are introduced to improve the scalability of GNNs. GraphSAGE \[11\] and PASS \[38\] sample fixed-size neighbors from the full neighborhood for each node in each step. FastGCN \[5\], Adaptive-GCN \[13\] and LADIES \[45\] sample the same amount of nodes in each propagation step. Alternatively, Cluster-GCN \[7\], GraphSAINT \[41\] and ShadowGNN \[40\] directly extract the subgraph containing the strongly correlated nodes around the anchor node. All these sampling methods do not consider the preserving of target entities as well as the different importance of entities for different queries.

**GNNs for KG reasoning.** The full propagation methods, such as R-GCN \[26\] and CompGCN \[30\], propagate over the full set of entities \(V\) with different message functions. These methods do not distinguish the importance of different entities for different queries, thus perform bad. GraIL \[28\] propagates within the enclosing subgraphs extracted between entities, which enables inductive reasoning but is very expensive to evaluate multiple candidate entities. Recently, NBFNet \[44\] and RED-GNN \[43\] progressively propagate from the query entity \(e_q\) to its \(\ell\)-hop neighbors in the \(\ell\)-th step. They show significance in both transductive and inductive reasoning and can efficiently evaluate multiple candidate entities. However, the amount of involved entities grows exponentially with deeper propagation path, making it harder to recognize the target answer entity. Overall, the existing GNN-based methods adopt hand-designed propagation schemes.

5 Conclusion

In this paper, we propose a general propagation framework for GNN-based KG reasoning and revisit the state-of-the-art methods from the lens of propagation path. By comparing different propagation schemes based on involved entities, target over candidates and per-hop performance, we observe that a deeper propagation depth can better capture the long-range information, but controlling the number of involved entities in deeper steps is important. Hence, we propose AdaProp to learn an adaptive propagation path that filters out irrelevant entities while preserving promising targets during propagation. The proposed method contains an incremental sampling mechanism to preserve the promising target and a learning-based sampling distribution to identify the promising entities. Empirical studies on several benchmarking datasets prove the superiority of AdaProp, namely it is efficient and achieves state-of-the-art performance in both transductive and inductive settings.
There are a few limitations for AdaProp. First, the straight-through estimator is a biased gradient estimator. A better design of the learning strategy may further improve the performance. Second, AdaProp is only evaluated on reasoning missing triplet. Reasoning on the multi-hop queries [24] is more complex. In the future, we would like to try the other learning strategies and apply AdaProp to more complex reasoning scenarios.

References

[1] P. W Battaglia, J. B Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, et al. Relational inductive biases, deep learning, and graph networks. Technical report, arXiv:1806.01261, 2018.
[2] Y. Bengio, N. Léonard, and A. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. Technical report, arXiv:1308.3432, 2013.
[3] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko. Translating embeddings for modeling multi-relational data. In NeurIPS, pages 2787–2795, 2013.
[4] Y. Cao, X. Wang, X. He, Z. Hu, and T. Chua. Unifying knowledge graph learning and recommendation: Towards a better understanding of user preferences. In TheWebConf, pages 151–161, 2019.
[5] J. Chen, T. Ma, and C. Xiao. Fastgcn: Fast learning with graph convolutional networks via importance sampling. In ICLR, 2018.
[6] X. Chen, S. Jia, and Y. Xiang. A review: Knowledge reasoning over knowledge graph. Expert Systems with Applications, 141:112948, 2020.
[7] W. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C. Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In SIGKDD, pages 257–266, 2019.
[8] R. Das, S. Dhuliawala, M. Zaheer, L. Vilnis, I. Durugkar, A. Krishnamurthy, A. Smola, and A. McCallum. Go for a walk and arrive at the answer: Reasoning over paths in knowledge bases using reinforcement learning. In ICLR, 2017.
[9] T. Dettmers, P. Minervini, P. Stenetorp, and S. Riedel. Convolutional 2D knowledge graph embeddings. In AAAI, 2017.
[10] J. Gilmer, S. Schoenholz, P. F Riley, O. Vinyals, and G. E Dahl. Neural message passing for quantum chemistry. In ICML, pages 1263–1272, 2017.
[11] W. Hamilton, Z. Ying, and J. Leskovec. Inductive representation learning on large graphs. In NeurIPS, pages 1024–1034, 2017.
[12] W. Hu, M. Fey, M. Zitnik, Y. Dong, H. Ren, B. Liu, M. Catasta, and J. Leskovec. Open graph benchmark: Datasets for machine learning on graphs. NeurIPS, 2020.
[13] W. Huang, T. Zhang, Y. Rong, and J. Huang. Adaptive sampling towards fast graph representation learning. In NeurIPS, volume 31, pages 4558–4567, 2018.
[14] X. Huang, J. Zhang, D. Li, and P. Li. Knowledge graph embedding based question answering. In WSDM, pages 105–113, 2019.
[15] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. In ICLR, 2017.
[16] S. Ji, S. Pan, E. Cambria, P. Marttinen, and P. S Yu. A survey on knowledge graphs: Representation, acquisition and applications. Technical report, arXiv:2002.00388, 2020.
[17] T. N Kipf and M. Welling. Semi-supervised classification with graph convolutional networks. In ICLR, 2016.
[18] W. Kool, H. Van Hoof, and M. Welling. Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement. In ICML, pages 3499–3508. PMLR, 2019.
[19] C. Meilicke, M. Fink, Y. Wang, D. Ruffinelli, R. Gemulla, and H. Stuckenschmidt. Fine-grained evaluation of rule-and embedding-based systems for knowledge graph completion. In ISWC, pages 3–20. Springer, 2018.
[20] A. Mnih and K. Gregor. Neural variational inference and learning in belief networks. In ICML, pages 1791–1799. PMLR, 2014.
[21] K. Oono and T. Suzuki. Graph neural networks exponentially lose expressive power for node classification. In ICLR, 2019.
[22] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. In ICLR, 2017.
[23] M. Qu, J. Chen, L. Xhonneux, Y. Bengio, and J. Tang. Rnnlogic: Learning logic rules for reasoning on knowledge graphs. In ICLR, 2021.
[24] H. Ren, W. Hu, and J. Leskovec. Query2box: Reasoning over knowledge graphs in vector space using box embeddings. In ICLR, 2019.
[25] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. In ICLR, 2017.
[26] M. Schlichtkrull, T. N Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling. Modeling relational data with graph convolutional networks. In ESWC, pages 593–607. Springer, 2018.
[27] R. S Sutton and A. Barto. Reinforcement learning: An introduction. MIT press, 2018.
[28] K. K Teru, E. Denis, and W. L Hamilton. Inductive relation prediction by subgraph reasoning. Technical report, arXiv:1911.06962, 2020.
[29] K. Toutanova and D. Chen. Observed versus latent features for knowledge base and text inference. In PWCVSMC, pages 57–66, 2015.
[30] S. Vashishth, S. Sanyal, V. Nitin, and P. Talukdar. Composition-based multi-relational graph convolutional networks. 2019.
[31] Q. Wang, Z. Mao, B. Wang, and L. Guo. Knowledge graph embedding: A survey of approaches and applications. TKDE, 29(12):2724–2743, 2017.
[32] X. Wang, X. He, Y. Cao, M. Liu, and T. Chua. Krag: Knowledge graph attention network for recommendation. In SIGKDD, pages 950–958, 2019.
[33] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning Journal, 8(3-4):229–256, 1992.
[34] S. Xie and S. Ermon. Reparameterizable subset sampling via continuous relaxations. In IJCAI, 2019.
[35] W. Xiong, T. Hoang, and W. Wang. Deeppath: A reinforcement learning method for knowledge graph reasoning. In EMNLP, pages 564–573, 2017.
[36] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. How powerful are graph neural networks? In ICLR, 2018.
[37] F. Yang, Z. Yang, and W. W Cohen. Differentiable learning of logical rules for knowledge base reasoning. In NeurIPS, pages 2319–2328, 2017.
[38] M. Yoon, T. Gervet, B. Shi, S. Niu, Q. He, and J. Yang. Performance-adaptive sampling strategy towards fast and accurate graph neural networks. In SIGKDD, pages 2046–2056, 2021.
[39] Y. Yu, K. Huang, C. Zhang, L. M Glass, J. Sun, and C. Xiao. Sumgmn: multi-typed drug interaction prediction via efficient knowledge graph summarization. Bioinformatics, 37(18):2988–2995, 2021.
[40] H. Zeng, M. Zhang, Y. Xia, A. Srivastava, A. Malevich, R. Kannan, V. Prasanna, L. Jin, and R. Chen. Decoupling the depth and scope of graph neural networks. In NeurIPS, volume 34, 2021.
[41] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna. Graphsaint: Graph sampling based inductive learning method. In ICLR, 2019.
[42] S. Zhang, Y. Tay, L. Yao, and Q. Liu. Quaternion knowledge graph embeddings. NeurIPS, 32, 2019.
[43] Y. Zhang and Q. Yao. Knowledge graph reasoning with relational directed graph. In TheWebConf, 2022.
[44] Z. Zhu, Z. Zhang, L. Xhonneux, and J. Tang. Neural bellman-ford networks: A general graph neural network framework for link prediction. In NeurIPS, 2021.
[45] D. Zou, Z. Hu, Y. Wang, S. Jiang, Y. Sun, and Q. Gu. Layer-dependent importance sampling for training deep and large graph convolutional networks. NeurIPS, 32:11249–11259, 2019.
Notations. In this paper, vectors are denoted by lowercase boldface, e.g., \( m, h, w \). Matrices are denoted by uppercase boldface, e.g., \( W \). Sets are denoted by script fonts, e.g., \( \mathcal{E}, \mathcal{V} \). We summarize the frequently used notations in Tab.1.

A knowledge graph is in the form of \( K = \{ \mathcal{V}, \mathcal{R}, \mathcal{E}, \mathcal{Q} \} \), where \( \mathcal{V}, \mathcal{R} \) are the sets of entities (nodes), relations (edge types), \( \mathcal{E} = \{ (e_s, r, e_o) | e_s, e_o \in \mathcal{V}, r \in \mathcal{R} \} \) is the set of triples (edges), and \( \mathcal{Q} = \{ (q, r, e_a) | q, e_a \in \mathcal{V}, r \in \mathcal{R} \} \) is the set of query triples. Given a query \( (q, r, ?) \), the reasoning task is to predict the answer entity \( e_a \) [6 38 37].

| notations | meanings |
|-----------|----------|
| \( \mathcal{V}, \mathcal{R}, \mathcal{E}, \mathcal{Q} \) | the set of entities, relations, facts (edges), queries |
| \( (e_s, r, e_o) \) | a fact triplet in \( \mathcal{E} \) with subject entity \( e_s \), relation \( r \) and object entity \( e_o \) |
| \( (e_q, r, e_a) \) | a query triple in \( \mathcal{Q} \) with query entity \( e_q \), query relation \( r \) and answer entity \( e_a \) |
| \( L \) | the total number of propagation steps |
| \( \ell \) | the \( \ell \)-th propagation step and \( \ell \in \{0 \ldots L\} \) |
| \( \mathcal{V}^\ell \) | the set of involved entities in the \( \ell \)-th propagation step |
| \( \mathcal{E}^\ell \) | the set of edges connecting entities in \( \mathcal{V}^{\ell-1} \) and \( \mathcal{V}^\ell \) |
| \( \mathcal{V}^\ell \) | the set of newly-visited entities in the \( \ell \)-th propagation step |
| \( m^\ell_{(e_s, r, e_o)} \) | the message on edge \( (e_s, r, e_o) \) at step \( \ell \) |
| \( h^\ell_e \) | the representation of entity \( e \) at step \( \ell \) |

Our code is added in the anonymous link: [https://anonymous.4open.science/r/AdaProp-NeurIPS2022/](https://anonymous.4open.science/r/AdaProp-NeurIPS2022/)

### A Details of works in the literature

#### A.1 MESS(·) and AGG(·) for GNN-based methods

Table 6: Summary of the GNN functions for message propagation. The values in \{ \} represent different operation choices that are tuned as hyper-parameters for different datasets.

| method | \( m^\ell_{(e_s, r, e_o)} := \text{MESS}(\cdot) \) | \( h^\ell_{e_o} := \text{AGG}(\cdot) \) |
|-------------------|--------------------------------------|--------------------------------------|
| R-GCN \[26\]       | \( W^\ell_r h^\ell_{e_s} \), where \( W_r \) depends on the relation \( r \) | \( W^\ell_r h^\ell_{e_s} + \sum_{e_o \in N(e_s)} \frac{1}{c_{o,r}} m^\ell_{(e_s, r, e_o)} \) |
| CompGCN \[30\]     | \( W^\ell_{\lambda(r)} (\cdot, \cdot, \cdot) \langle h_{e_s}^{\ell-1}, h^\ell_e \rangle \), where \( W^\ell_{\lambda(r)} \) depends on the direction of \( r \) | \( \sum_{e_o \in N(e_s)} m^\ell_{(e_s, r, e_o)} \) |
| GralL \[38\]       | \( \alpha^\ell_{(e_s, r, e_o)} r_q (W^\ell_r h^\ell_{e_s} + W^\ell_r h^\ell_{e_o}) \), where \( \alpha^\ell_{(e_s, r, e_o)} \) depends on the direction of \( r \) | \( W^\ell_r h^\ell_{e_s} + \sum_{e_o \in N(e_s)} \frac{1}{c_{o,r}} m^\ell_{(e_s, r, e_o)} \) |
| NBFNet \[40\]      | \( W^\ell_{\{+, \cdot, \cdot\}} \langle h_{e_s}^{\ell-1}, w_q(r, r_q) \rangle \), where \( w_q(r, r_q) \) is a query-dependent weight vector | \{Sum, Mean, Max, PNA\}_{e_o \in N(e_s)} m^\ell_{(e_s, r, e_o)} \) |
| RED-GNN \[43\]     | \( \alpha^\ell_{(e_s, r, e_o)} r_q \langle h_{e_s}^{\ell-1} + h^\ell_e \rangle \), where \( \alpha^\ell_{(e_s, r, e_o)} \) is the attention weight | \( \sum_{e_o \in N(e_s)} m^\ell_{(e_s, r, e_o)} \) |
| AdaProp \[44\]     | \( \alpha^\ell_{(e_s, r, e_o)} r_q \langle +, \cdot, \cdot\rangle \langle h_{e_s}^{\ell-1} + h^\ell_e \rangle \), where \( \alpha^\ell_{(e_s, r, e_o)} \) is as in \[43\] | \{Sum, Mean, Max\}_{e_o \in N(e_s)} m^\ell_{(e_s, r, e_o)} \) |

The two functions of the GNN-based methods are summarized in Tab.6. The main differences of different methods are the combiners for entity and relation representations on the edges, the operators on the different representations, and the attention weights. Recent works \[30 40\] have
shown that the different combiners and operators only have slight influence on the performance. Comparing GraIL and RED-GNN, even though the message functions and aggregation functions are similar, their empirical performances are quite different with different propagation patterns. Hence, the design of propagation path is the key factors influencing the performance of different methods.

A.2 Discussion on the constrained propagation GraIL [28]

In KG reasoning, the key objective is to find the target answer entity \( e_a \) given \( (e_q, r_q, ?) \). To fulfill this goal, the model is required to evaluate multiple candidate answer entities. Since all the entities in the KG can be the candidates, we need to evaluate \(|V|\) entities for a single query \((e_q, r_q, ?)\).

For full propagation methods, the high-level embeddings of all the entities can be obtained in a single forward pass. Then the evaluation on all the candidate answers can be efficient since computing the score based on high-level embedding is cheap. For progressive propagation, the candidates can also be evaluated in a single forward pass as the last step representations are directly used to score the candidates.

However, for the constrained propagation method GraIL, the propagation path depends on both the query entity \( e_q \) and the answer entity \( e_a \), i.e., \( V^f = V^f_{(e_q, e_a)} \). Hence, it requires \(|V|\) forward passes to evaluate all the entities, which is extremely expensive when the number of entities is large. For transductive reasoning, all the datasets have tens of thousands of entities. Running GraIL on them is intractable.

B Supplementary materials to the sampling technique

B.1 Details of the Gumbel top-\( k \) metric

Inspired by the recent works [18, 34] in sampling subsets from the categorical distribution, we adopt the Gumbel top-\( k \) trick to sample from the distribution. Given the distribution \( p^f(e) = \frac{\exp(\phi_e / \tau)}{\sum_{e \in \mathcal{V}} \exp(\phi_e / \tau)} \) with logits \( \phi_e = g(h_e^f; \theta^f) \) and temperature value \( \tau > 0 \), the Gumbel-tricks sample \( N = |\mathcal{V}| \) independent noises from the uniform distribution, i.e., \( U_e \sim \text{Uniform}(0, 1) \) to form the Gumbel logits \( G_e = \phi_e - \log(-\log U_e) \). Rather than directly sample from distribution \( p(e) \), we connect the top-\( K \) entities in \( \mathcal{V} \) based on the values of Gumbel logits \( G_e \). As proved in [18, 34], this procedure is equivalent to sample without replacement from \( p(e) \).

B.2 Straight-through estimator

Considering the discrete nature of the propagation path \( \hat{\mathcal{L}}(\theta) \), the direct gradient of loss function \( \mathcal{L} \) w.r.t. the sampling parameters \( \theta \) is intractable. The reparameterization technique for Gumbel distributions [13] does not do explicit sampling, thus is not appropriate for the case here. Instead, we choose the straight-through (ST) estimator [2, 15], which can approximately estimate gradient for discrete variables. The key idea is to back-propagate through the sampling signal as if it was the identity function. Given the sampled entities \( e_o \in \mathcal{V}^f \) as well as their sampling probability \( p^f(e_o) \), the derivative of the sampling parameters can be obtained by multiplying \( p^f(e_o) \) with the hidden states \( h^f_{e_o} \). Specifically, rather than directly use the entity representations \( h^f_{e_o} \) to compute messages, we use

\[
\begin{align*}
    h^f_{e_o} &:= (1 - \text{no\_grad}(p^f(e_o))) + p^f(e_o)) \cdot h^f_{e_o},
\end{align*}
\]

where \( \text{no\_grad}(p^f(e_o)) \) means that the back-propagation signals will not go through this term. In this approach, the forward computation will not be influenced, while the backward signals can go through the multiplier \( p^f(e_o) \). Besides, the backward signals, even though not exactly equivalent to the true gradient, is in positive propagation to it. Hence, we can use such a backward signal to approximately estimate the gradient of \( \mathcal{L} \) w.r.t. \( \theta \).

B.3 Adapting the sampling methods for KG reasoning

When modeling the homogeneous graphs, the sampling methods are designed to solve the scalability problem. Besides, they are mainly used in the node classification task, which is quite different from
the KG reasoning task here. Hence, we introduce the adaptations we made to use the sampling methods designed for homogeneous graphs in the KG reasoning scenario.

- **Node-wise sampling.** The design in this type is based on GraphSAGE [11]. Given $V^{\ell-1}$, we sample $K$ neighboring entities from $N(e)$ for entities $e \in V^{\ell-1}$. Then the union of these sampled entities forms $V^\ell$ for next step propagation. For the not-learned version, we randomly sample from $N(e)$. For the learned version, we sample from $N(e)$ based on the distribution $p^\ell(e) \sim g(h^*_e; \theta)$.

- **Layer-wise sampling.** The design in this type is based on LADIES [45]. Given $V^{\ell-1}$, we firstly collect the set of neighboring entities $V^{\ell-1}_{\text{neib}} = \bigcup N(e), e \in V^{\ell-1}$. Then, we sample $K$ entities without replacement from $V^{\ell-1}_{\text{neib}}$ as $V^\ell$ for next step propagation. For the not-learned version, we sample according to the degree distribution for entities in $V^{\ell-1}_{\text{neib}}$. For the learned version, we sample from $V^{\ell-1}_{\text{neib}}$ based on the distribution $p^\ell(e) \sim g(h^*_e; \theta)$.

- **Subgraph sampling.** Given the query $(e_q, r_q, ?)$, we generate multiple random walks starting from the query entity $e_q$ and use the sampled entities to induce the subgraph $V_{e_q}$. Then, the propagation starts from $e_q$ like the progressive propagation, but it is constrained within the range of $V_{e_q}$. Since learning to sample the subgraph is challenging, we only evaluate the subgraph sampling in the not-learned version.

### B.4 Comparison of straight-through estimator with REINFORCE estimator

The ST estimator uses the non-differentiable discrete samples in the forward pass, but the continuous values are used in the backward pass. In addition to ST, another common gradient estimators for discrete variables is the REINFORCE [33, 20], where the gradients are back-propagated by the log-trick.

In this part, we empirically compare the straight-through (ST) estimator and the REINFORCE estimator. We show the learning curves of the two gradient estimators on WN18RR and FB15k237-v1 in Fig[7]. The random sampler without optimizer is included as a reference baseline. We have the following observations:

- Both gradient estimators can learn informative signals and outperform the not-learned random sampler.
- The REINFORCE estimator is known to have high variance [27] and such a high variance will be accumulated across propagation steps, leading to unstable learning curve.
- The ST estimator is significantly better than the REINFORCE estimator with better any-time performance.

![Optimization methods for the sampler](image-url)

Figure 7: Optimization methods for the sampler.
C Supplementary materials to experiments

C.1 Datasets

We show the statistics of transductive datasets in Tab.7. All of the three datasets have tens of thousands of entities. CompGCN [30], which propagates within the full KG \( \mathcal{V} \), will easily run out-of-memory, on the larger KG NELL-995.

Table 7: Statistics of the transductive KG datasets. “fact” denotes the number of triples used as graph, i.e., \(|\mathcal{G}|\), and “tra”, “val” and “tst” are the triplets used for reasoning, i.e., \(|\mathcal{Q}_{\text{tra}}|, |\mathcal{Q}_{\text{val}}|, |\mathcal{Q}_{\text{tst}}|\), respectively.

| dataset     | #entity | #relation | fact   | tra     | val     | test    |
|-------------|---------|-----------|--------|---------|---------|---------|
| WN18RR      | 40.9k   | 11        | 65.1k  | 21.7k   | 3.0k    | 3.1k    |
| FB15k-237   | 14.5k   | 237       | 204.1k | 68.0k   | 17.5k   | 20.4k   |
| NELL-995    | 74.5k   | 200       | 112.2k | 37.4k   | 543     | 2.8k    |

We show the statistics of inductive datasets in Tab.8. The scale of these datasets are much smaller than those in Tab.7 as the datasets here are just subsets. Even though the scale is smaller, we show that deeper propagation path is important as well.

Table 8: Statistics of the inductive datasets. “fact” denotes the number of triplets used as graph, i.e., \(|\mathcal{G}|\), and “pred” presents the triplets used for reasoning, i.e., \(|\mathcal{Q}|\)

| datasets | WN18RR | FB15k-237 | NELL-995 |
|----------|--------|-----------|----------|
|          | ent    | rel       | fact     | ent      | rel       | fact     | ent      | rel       | fact     |
| v1 train | 2746   | 9         | 5410     | 1268     | 1594      | 180      | 4245     | 981       | 3102      | 14       |
| v1 test  | 922    | 1618      | 373      |          | 1093      | 1993     | 411      |           | 255       | 833      |
| v2 train | 6954   | 10        | 15262    | 3706     | 1608      | 200      | 9739     | 2346      | 2563      | 88       |
| v2 test  | 5084   | 4011      | 852      |          | 1660      | 4145     | 947      |           | 2086      | 4586     |
| v3 train | 12078  | 11        | 25901    | 6249     | 3668      | 215      | 17986    | 4408      | 4647      | 142      |
| v3 test  | 5084   | 6327      | 1140     |          | 2501      | 7406     | 1731     |           | 3566      | 8048     |
| v4 train | 3861   | 9         | 7940     | 1902     | 4707      | 219      | 27203    | 6713      | 2092      | 76       |
| v4 test  | 7084   | 12334     | 2823     |          | 3051      | 11714    | 2840     |           | 2795      | 7073     |

C.2 Hyper-parameters

We tune the number of propagation steps \( L \) among \( \{5, 6, 7, 8\} \), the number of incrementally sampled entities \( K \) among \( \{100, 500, 1000, 2000\} \), temperature \( \tau \) in \( \{0.5, 1.0, 2.0\} \), batch size in \( \{20, 50, 100\} \), \( \text{MESS}(-) \) in \( \{+ , * , \circ\} \) and \( \text{AGG}(-) \) in \{Sum, Mean, Max\} The ranges of the other hyper-parameters are kept the same as RED-GNN. In the transductive setting, we select the optimal hyper-parameters for evaluation based on the MRR metric on the validation set. In the inductive setting, we select based on the Hit@10 metric on the validation set.

We provide the hyper-parameters of \( L, K, \tau \) and batch size in Tab.9 and Tab.10 for transductive and inductive setting, respectively. We observe that

• In most of the datasets, we have a larger number of propagation steps with \( L \geq 6 \).
• In the transductive setting, the values of \( K \) are larger since the KGs are larger. But in the inductive setting, \( K \)’s are generally small. This means that \( K \) is not as large as possible. More irrelevant entities will influence the performance.
• There is no much regularity in the choices of \( \tau \).
• The sum aggregator is always the best.
Table 9: Optimal hyper-parameters for the transductive datasets.

| Hyper-parameters | WN18RR | FB15k237 | NELL-995 |
|------------------|--------|----------|----------|
| \( L \)          | 8      | 7        | 6        |
| \( K \)          | 1000   | 2000     | 2000     |
| \( \tau \)       | 0.5    | 2.0      | 0.5      |
| batch size       | 100    | 50       | 20       |
| MESS(\cdot)      | *      | +        | *        |
| AGG(\cdot)       | SUM    | SUM      | SUM      |

Table 10: Optimal hyper-parameters for the inductive datasets.

| Hyper-parameters | V1 | V2 | V3 | V4 | V1 | V2 | V3 | V4 | V1 | V2 | V3 | V4 |
|------------------|----|----|----|----|----|----|----|----|----|----|----|----|
| \( L \)          | 7  | 7  | 7  | 6  | 5  | 5  | 7  | 7  | 6  | 5  | 8  | 5  |
| \( K \)          | 200| 100| 200| 200| 300| 200| 200| 500| 200| 200| 500| 100|
| \( \tau \)       | 0.5| 0.5| 1.0| 1.0| 0.5| 2.0| 1.0| 1.0| 0.5| 1.0| 1.0| 2.0|
| batch size       | 20 | 20 | 50 | 50 | 50 | 20 | 20 | 50 | 20 | 50 | 20 | 50 |
| MESS(\cdot)      | +  | +  | +  | +  | +  | +  | +  | +  | +  | +  | +  | +  |
| AGG(\cdot)       | SUM| SUM| SUM| SUM| SUM| SUM| SUM| SUM| SUM| SUM| SUM| SUM|

C.3 Properties for different propagation depth \( L \)

Same as Fig 2(a), we show the ratio of involved entities \( \text{IE}(\hat{G}^L) \), target over candidates \( \text{ToC}(\hat{G}^L) \), per-hop performance w.r.t. different propagation depths, as well as the distance distribution of FB15k237 and NELL-995 here for transductive reasoning. Since the full propagation methods do not work in the inductive setting, we mainly compare the three propagation types in transductive setting. As CompGCN will easily run out of memory on NELL-995, we can only provide results on \( L = 1 \).

![Figure 8: Values for different propagation depth \( L \) on FB15k-237 and NELL-995.](image)

The observations are the same as those in Fig 2(a) Namely, most of the target answer entities are close to the queried one, and it is important for the propagation path to go deeper such that the long-range information can be captured.
C.4 Full results of inductive reasoning

The full results of inductive reasoning, for MRR, Hit@1 and Hit@10 metrics, are shown in Table 11. In the other metrics, i.e., MRR and Hit@1, AdaProp is also the best in most of the cases.

Table 11: Inductive setting. Best performance is indicated by the bold face numbers, and the underline means the second best.

| metrics     | methods  | V1     | V2     | V3     | V4     | V1     | V2     | V3     | V4     | V1     | V2     | V3     | V4     |
|-------------|----------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
|             | WN18RR   |        |        |        |        |        |        |        |        |        |        |        |        |
|             | RuleN    | 0.668  | 0.645 | 0.368 | 0.624  | 0.363  | 0.433  | 0.439 | 0.429  | 0.615  | 0.385  | 0.381  | 0.333  |
|             | Neural LP| 0.649  | 0.635 | 0.361 | 0.628  | 0.325  | 0.389  | 0.400 | 0.396  | 0.610  | 0.361  | 0.367  | 0.261  |
|             | DRUM     | 0.666  | 0.646 | 0.380 | 0.627  | 0.333  | 0.395  | 0.402 | 0.410  | 0.628  | 0.365  | 0.375  | 0.273  |
| MRR         | GraIL    | 0.627  | 0.625 | 0.323 | 0.553  | 0.279  | 0.276  | 0.251 | 0.227  | 0.481  | 0.297  | 0.322  | 0.262  |
|             | RED-GNN  | 0.701  | 0.690 | 0.427 | 0.651  | 0.369  | 0.469  | 0.445 | 0.442  | 0.637  | 0.419  | 0.436  | 0.363  |
|             | NBFNet   | 0.371  | 0.351 | 0.238 | 0.317  | 0.186  | 0.217  | 0.186 | 0.183  | 0.275  | 0.221  | 0.228  | 0.149  |
|             | AdaProp  | 0.733  | 0.715 | 0.474 | 0.662  | 0.310  | 0.471  | 0.471 | 0.454  | 0.644  | 0.452  | 0.435  | 0.366  |
|             | FB15k237 |        |        |        |        |        |        |        |        |        |        |        |        |
|             | RuleN    | 0.598  | 0.570 | 0.281 | 0.430  | 0.220  | 0.216  | 0.216 | 0.216  | 0.348  | 0.270  | 0.270  | 0.270  |
|             | Neural LP| 0.560  | 0.541 | 0.238 | 0.317  | 0.180  | 0.211  | 0.180 | 0.180  | 0.271  | 0.220  | 0.220  | 0.220  |
|             | DRUM     | 0.580  | 0.560 | 0.292 | 0.409  | 0.219  | 0.219  | 0.219 | 0.219  | 0.371  | 0.299  | 0.299  | 0.299  |
|             | GraIL    | 0.554  | 0.542 | 0.278 | 0.443  | 0.205  | 0.205  | 0.205 | 0.205  | 0.425  | 0.299  | 0.299  | 0.299  |
|             | RED-GNN  | 0.653  | 0.633 | 0.368 | 0.606  | 0.302  | 0.381  | 0.351 | 0.340  | 0.525  | 0.319  | 0.345  | 0.259  |
|             | NBFNet   | -      | -      | -      | -      | 0.000  | 0.000  | 0.000 | 0.000  | 0.000  | 0.000  | 0.000  | 0.000  |
|             | AdaProp  | 0.668  | 0.642 | 0.396 | 0.611  | 0.191  | 0.372  | 0.377 | 0.353  | 0.522  | 0.344  | 0.337  | 0.247  |
|             | NELL-995 |        |        |        |        |        |        |        |        |        |        |        |        |
|             | RuleN    | 0.573  | 0.547 | 0.288 | 0.424  | 0.206  | 0.206  | 0.206 | 0.206  | 0.362  | 0.288  | 0.288  | 0.288  |
|             | Neural LP| 0.532  | 0.509 | 0.260 | 0.339  | 0.180  | 0.211  | 0.180 | 0.180  | 0.271  | 0.220  | 0.220  | 0.220  |
|             | DRUM     | 0.552  | 0.531 | 0.315 | 0.408  | 0.219  | 0.219  | 0.219 | 0.219  | 0.371  | 0.299  | 0.299  | 0.299  |
|             | GraIL    | 0.554  | 0.542 | 0.278 | 0.443  | 0.205  | 0.205  | 0.205 | 0.205  | 0.425  | 0.299  | 0.299  | 0.299  |
|             | RED-GNN  | 0.653  | 0.633 | 0.368 | 0.606  | 0.302  | 0.381  | 0.351 | 0.340  | 0.525  | 0.319  | 0.345  | 0.259  |
|             | NBFNet   | -      | -      | -      | -      | 0.000  | 0.000  | 0.000 | 0.000  | 0.000  | 0.000  | 0.000  | 0.000  |
|             | AdaProp  | 0.668  | 0.642 | 0.396 | 0.611  | 0.191  | 0.372  | 0.377 | 0.353  | 0.522  | 0.344  | 0.337  | 0.247  |

C.5 The more visualization results

We provide more visualization results in Fig. 9. In each line, we show the queries $q_1$ and $q_2$ with the same query entity but different query relations. There is a clear different between the two propagation paths of $q_1$ and $q_2$ learned by AdaProp are quite different. But they are the same for the progressive propagation method RED-GNN, where only the edge weights are different. This demonstrate that AdaProp learns query dependent propagation path.
Figure 9: More examples of the propagation paths.
The more ablation studies

In Section 3.2, we have evaluated the influence of different propagation depth \( L \). Here, we show the other two factors related to the sampling distribution, i.e., the number of sampled entities \( K \) and the temperature value \( \tau \).

We show the influence of different values of \( K \) in Fig. 10. The different propagation depths \( L \) is also included in this figure. As shown, the larger \( K \) can lead to the better performance. However, the performance gains by sampling more entities would gradually become marginal, especially for larger propagation depth. It means that the more irrelevant information to the queries will be included with a larger \( K \). Besides, selecting a larger \( K \) will increase the training cost. Hence, it is better to balance this trade-off and choose a moderate value of \( K \).

We show the performances of different temperatures in Tab. 12. The influence of different choices of \( \tau \) is not high on the different datasets. This means that the sampling signal is stable and it can be adjusted by the sampling weights as well.

| \( \tau \) | WN18RR  | FB15k237-v1 |
|-----------|---------|-------------|
| 0.5       | 0.562   | 0.551       |
| 1.0       | 0.559   | 0.546       |
| 2.0       | 0.560   | 0.549       |